Comparative Analysis of Robust Design Methods

by

Jagmeet Singh

B. Tech., Mechanical Engineering
Indian Institute of Technology at Kanpur, 2001

S.M., Mechanical Engineering
Massachusetts Institute of Technology, 2003

Submitted to the Department of Mechanical Engineering
In Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

© 2006 Massachusetts Institute of Technology
All rights reserved

Signature of Author_____
Department of Mechanical Engineering
May 5, 2006

Certified by_____
Daniel D. Frey
Assistant Professor, Department of Mechanical Engineering and Engineering Systems
Thesis Supervisor

Accepted by_____
Lallit Anand
Chairman, Department Committee on Graduate Studies

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 14 2006
LIBRARIES

BARKER

Comparative Analysis of Robust Design Methods

by

Jagmeet Singh

Submitted to Department of Mechanical Engineering
on May 5, 2006 in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Mechanical Engineering

Abstract

Robust parameter design is an engineering methodology intended as a cost effective approach to improve the quality of products, processes and systems. Control factors are those system parameters that can be easily controlled and manipulated. Noise factors are those system parameters that are difficult and/or costly to control and are presumed uncontrollable. Robust parameter design involves choosing optimal levels of the controllable factors in order to obtain a target or optimal response with minimal variation. Noise factors bring variability into the system, thus affecting the response. The aim is to properly choose the levels of control factors so that the process is robust or insensitive to the variation caused by noise factors. Robust parameter design methods are used to make systems more reliable and robust to incoming variations in environmental effects, manufacturing processes and customer usage patterns. However, robust design can become expensive, time consuming, and/or resource intensive. Thus research that makes robust design less resource intensive and requires less number of experimental runs is of great value. Robust design methodology can be expressed as multi-response optimization problem. The objective functions of the problem being: maximizing reliability and robustness of systems, minimizing the information and/or resources required for robust design methodology, and minimizing the number of experimental runs needed. This thesis discusses various noise factor strategies which aim to reduce number of experimental runs needed to improve quality of system. Compound Noise and Take-The-Best-Few Noise Factors Strategy are such noise factor strategies which reduce experimental effort needed to improve reliability of systems. Compound Noise is made by combing all the different noise factors together, irrespective of the number of noise factors. But such a noise strategy works only for the systems which show effect sparsity. To apply the Take-The-Best-Few Noise Factors Strategy most important noise factors in system's noise factor space are found. Noise factors having significant impact on system response variation are considered important. Once the important noise factors are

identified, they are kept independent in the noise factor array. By selecting the few most important noise factors for a given system, run size of experiment is minimized. Take-The-Best-Few Noise Factors Strategy is very effective for all kinds of systems irrespective of their effect sparsity. Generally Take-The-Best-Few Noise Factors Strategy achieves nearly 80% of the possible improvement for all systems. This thesis also tries to find the influence of correlation and variance of induced noise on quality of system. For systems that do not contain any significant three-factor interactions correlation among noise factors can be neglected. Hence amount of information needed to improve the quality of systems is reduced.

Thesis Supervisor: Daniel D. Frey
Title: Robert N. Noyce Assistant Professor, Department of Mechanical Engineering & Engineering Systems

Committee Member: Daniel E. Whitney
Title: Senior Research Scientist, Center for Technology, Policy and Industrial Development and Department of Mechanical Engineering, Senior Lecturer in Engineering Systems

Committee Member: Warren P Seering
Title: Weber-Shaughness Professor, Department of Mechanical Engineering

Acknowledgements

I offer prayer of thanks to GOD for giving me this wonderful opportunity to do research in one of the premier institutes. I would like to thank my thesis advisor Dan Frey. I am always short of words for his mentorship, support, guidance, patience and integrity. I have learnt a lot from him in both research and life. He is one of best advisors that one could have asked for in research. It was for him that I truly enjoyed my PhD research and was always excited about it through out my duration of PhD.

Dan Whitney and Warren Seering are the best committee members to have. Their keen observations and tremendous experience helped me a great deal in shaping path of my research. They helped me a lot in putting my research results in perspective. I have been learning from them since the day I joined MIT. I again find myself short of words to thank both of them for their support, guidance, mentorship and blessings. I would also like to thanks Nathan Soderborg, Joe Saleh and Ford-MIT Research Alliance for supporting this research.

My grandparents, parents, Mr. Yashpal Singh and Mrs. Surjeet Kaur and sister, Jaideep Kaur provided me with support all throughout my research. And no acknowledgement is complete without mentioning Rajesh Jugulum. He is part of our research group. His reviews on research progress, his insights were very helpful in the progress of research. I would also like to thank my friends for their goodwill and support.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Motivation

Robust parameter design is an engineering methodology intended as a cost effective approach to improve the quality of products, processes and systems, Taguchi (1987), Robinson et al. (2004). Taguchi (1987) proposed that inputs to any system can be classified as control factors and noise factors. Control factors are those system parameters that can be easily controlled and manipulated. Noise factors are those system parameters that are difficult and/or costly to control and are presumed uncontrollable. Robust parameter design involves choosing optimal levels of the controllable factors in order to obtain a target or optimal response with minimal variation. The challenge arises in obtaining optimal response due to the influence of the uncontrollable noise factors. Noise factors bring variability into the system, thus affecting the response. The aim is to properly choose the levels of control factors so that the process is robust or insensitive to the variation caused by noise factors.

Robust parameter design is among one of the most important developments in systems engineering in 20^{th} century, Clausing and Frey (2005). These methods seemed to have accounted for a significant part of quality differential that made Japanese manufacturing dominant during 1970s. Robust parameter design enables in smoother system integration, faster transition to production, and higher field reliability.

Taguchi (1987) also proposed techniques of experimental design to identify the settings of control factors that would achieve robust performance of systems. He used orthogonal designs where an orthogonal array involving control factors ('inner array') is crossed with an orthogonal array involving noise factors ('outer array'). The response of the systems' at each setting of control factors were treated as replicates for the formulation of a measure that would be indicative of both the mean and variance of response. One of the weaknesses of these crossed array experiments is that they tend to require large number of experimental runs.

1.2 Goal of Research

Robust parameter design methods are used to make systems more reliable and robust to incoming variations in environmental effects, manufacturing processes and customer usage patterns. However, robust design can become expensive, time consuming, and/or resource intensive. Thus research that makes robust design less resource intensive and requires less number of experimental runs is of great value. Robust design methodology can be expressed as multi-response optimization problem. The objective functions of the problem being: maximizing reliability and robustness of systems, minimizing the information and/or resources required for robust design methodology, and minimizing the number of experimental runs needed. We will present noise strategies for robust design methods which would reduce the amount of experimental effort needed and information about noise factors space needed to maximize quality of a system.

1.3 Organization of Thesis

The thesis will first present simplest of the noise strategy which minimizes amount of experimental effort needed. Then it will suggest some alternative, useful and more efficient noise factor strategies. Chapter 2 will discuss the formulation of hierarchical probability model. This will form the basis to compare different robust design methods statistically. First the regularities exhibited by engineering systems will be discussed. Next those regularities will be put in a mathematical format. The mathematical formulation will be used to generate response surface instances to analyze different robust design methods. We will also discuss about selection of various parameters for hierarchical probability model.

Chapter 3 will introduce Compound Noise. Compound Noise is very effective as a robust design strategy on the systems which show *effect sparsity*. We will run two formulations of compound noise on response surface instances generated using strong and weak hierarchical probability model. Next these formulations of compound noise will be run on six different case studies from various engineering domains to verify conclusions from hierarchical probability model. In the end conditions for compound noise to be completely effectiveare outlined . We will also device an algorithm to use compound noise as a robust design method.

Chapter 4 will introduce Take-The-Best-Few Noise Factor Strategy, which is very effective as a robust design strategy for all kinds of systems. We will run this noise strategy on response surface instances generated using strong and weak hierarchical

probability model and six different case studies from various engineering domains. We will also compare this noise strategy with compound noise strategy. We will propose hybrid noise strategy as amalgamation of two noise strategies. We will then device an algorithm on the use of this noise strategy and compound noise strategy as robust design methods.

Chapter 5 will explore the influence of correlation among noise factors on robust design methods. We will see the impact of correlation and variance of induced noise factors on response surface instances and six different case studies. We will see that if system does not have any significant three-factor interaction then during robust design experiments we can neglect correlation and/or exaggerate intensity of induced noise factors. We will design an algorithm for implementing correlation influence in practice.

Chapter 6 will summarize the key messages from this thesis. It will present cost-benefit analysis of various robust design methods and the percentage improvement each robust design method can give for a given system. This cost-benefit analysis can be used by engineers to find the maximum benefit they can get out of a robust design study based on their allocated budget. Also it outlines scope of future research in area of robust design. This will be followed by references and appendices.

Chapter 2: Hierarchical Probability Model

2.1 Regularities in Engineering Systems

Experimentation is an important activity in design on systems. Almost every existing engineering system was shaped by a process of experimentation including preliminary investigation of phenomenon, sub-system prototyping, and system verification tests. Based on experience in planning and analyzing many experiments, practitioners and researchers in system design have identified regularities in the inter-relationships among factor effects and their interactions, Wu and Hamada (2000). Hamada and Wu (1992), Box and Meyer (1986), Chipman, Hamada and Wu (1997) and Wu and Hamada (2000) describe these regularities in detail:

- *Effect Sparsity Principle* – among many effects examined in any system only a small fraction of those effects are significant in system, Box and Meyer (1986). This is sometimes called the *Pareto principle in Experimental Design* based on analogy with the observations of the 19th century economist Vilfredo Pareto who argued that, in all countries and times, the distribution of income and wealth follows a logarithmic pattern resulting in the concentration of resources in the hands of a small number of wealthy individuals. Effect sparsity appears to be a phenomenon characterizing the knowledge of the experimenters more so than the physical or logical behavior of the system under investigation. Investigating an effect through experimentation requires an allocation of resources -- to resolve

more effects typically requires more experiments. Therefore, effect sparsity is in some sense an indication of wasted resources. If the important factor effects could be identified during planning, then those effects might be investigated exclusively, resources might be saved, and only significant effects would be revealed in the analysis.

- *Hierarchical Ordering Principle* – main effects are generally more significant that two factor interactions, two-factor interactions are generally more significant than three-factor interactions, and so on, Hamada and Wu (1992). Effects of same order are likely to have same significance level. This principle is also sometimes referred as "hierarchy". Effect hierarchy is illustrated in figure 2.1 for a system with four factors A, B, C and D. Figure 2.1illustrates a case in which hierarchy is not strict – for example, that some interactions (such as the two-factor interaction AC) are larger than some main effects (such as the main effect of B).

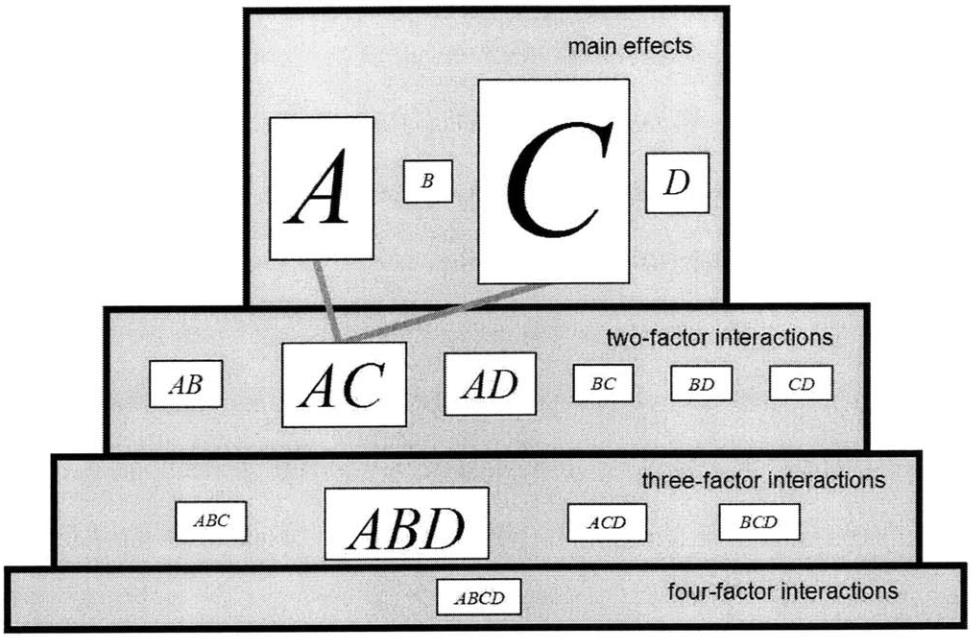


Figure 2.1: The hierarchy and heredity among main effects and interactions in a system with four factors *A*, *B*, *C*, and *D*. Interactions between factors are represented by letter combinations such as the two-factor interaction *AC*. The font size represents the size of the effect.

- *Effect Heredity Principle* – an interaction effect is likely to be significant when at least one of its parent factors is significant, Wu and Hamada (2000). It is also sometimes referred to as "inheritance".

2.2 Hierarchical Probability Model

Hierarchical probability models have been proposed as a means to analyze the results of experiments with complex aliasing patterns. In any system main effects and interaction effects present are of interest. There is also a need to predict the relative importance and

relationship among these effects. Chipman, Hamada and Wu (1997) have expressed these properties in mathematical form in a hierarchical prior probability model. The hierarchical probability model proposed by Chipman, Hamada and Wu (1997) has been extended here to enable evaluation of noise strategies in robust design. The model includes both control and noise factors since they are both needed for the present purposes. The model includes two-factor interactions since control by noise interactions are required for robust design to be effective. It also includes the possibility of three-factor interactions since these have been shown to be frequently present, especially in systems with a large number of factors Li and Frey (2005) and might affect the outcomes of robust design. The details of the model are in Equations 2.1 through 2.10.

$$y(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \beta_{ij} x_i x_j + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \sum_{\substack{k=1 \\ k>j}}^{n} \beta_{ijk} x_i x_j x_k + \varepsilon \qquad \textbf{(2. 1)}$$

$$x_i \sim NID(0, w_1^{\,2}) \;\; i \in 1 \ldots m \qquad \textbf{(2. 2)}$$

$$x_i \in \{+1, -1\} \;\; i \in m+1 \ldots n \qquad \textbf{(2. 3)}$$

$$\varepsilon \sim NID(0, w_2^{\,2}) \qquad \textbf{(2. 4)}$$

$$f(\beta_i | \delta_i) = \begin{cases} N(0,1) & \text{if } \delta_i = 0 \\ N(0, c^2) & \text{if } \delta_i = 1 \end{cases} \qquad \textbf{(2. 5)}$$

$$f(\beta_{ij}|\delta_{ij}) = \begin{cases} N(0, s_1^2) & \text{if } \delta_{ij} = 0 \\ N(0, c^2 \cdot s_1^2) & \text{if } \delta_{ij} = 1 \end{cases} \quad \textbf{(2. 6)}$$

$$f(\beta_{ijk}|\delta_{ijk}) = \begin{cases} N(0, s_2^2) & \text{if } \delta_{ijk} = 0 \\ N(0, c^2 \cdot s_2^2) & \text{if } \delta_{ijk} = 1 \end{cases} \quad \textbf{(2. 7)}$$

$$\Pr(\delta_i = 1) = p \quad \textbf{(2. 8)}$$

$$\Pr(\delta_{ij} = 1|\delta_i, \delta_j) = \begin{cases} p_{00} & \text{if } \delta_i + \delta_j = 0 \\ p_{01} & \text{if } \delta_i + \delta_j = 1 \\ p_{11} & \text{if } \delta_i + \delta_j = 2 \end{cases} \quad \textbf{(2. 9)}$$

$$\Pr(\delta_{ijk} = 1|\delta_i, \delta_j, \delta_k) = \begin{cases} p_{000} & \text{if } \delta_i + \delta_j + \delta_k = 0 \\ p_{001} & \text{if } \delta_i + \delta_j + \delta_k = 1 \\ p_{011} & \text{if } \delta_i + \delta_j + \delta_k = 2 \\ p_{111} & \text{if } \delta_i + \delta_j + \delta_k = 3 \end{cases} \quad \textbf{(2. 10)}$$

This hierarchical probability model allows any desired number of response surface instances to be created such that population of response surface instances has the desired properties of sparsity of effects, hierarchy, and inheritance. Equation 2.1 represents the measured response of the engineering system *y*. The independent variables x_i's are both control factors and noise factors. Control and noise factors are not distinguished in this notation except via indices. Equation 2.2 shows that the first set of *x* variables (x_1, x_2, ...

x_m) are regarded as "noise factors" and are assumed to be normally distributed. Equation 2.3 shows that the other x independent variables (x_{m+1}, x_{m+2}, ... x_n) are the "control factors" which are assumed to be two level factors. The variable represents the pure experimental error in the observation of the response which is assumed to be normally distributed. Since the control factors are usually explored over a wide range compared to the noise factors, the parameter w_1 is included to set the ratio of the control factors range to the standard deviation of the noise factors. The intensity of noise factors can be changed by changing the value of parameter w_1. The parameter w_2 is included to set the ratio of the standard deviation of the pure experimental error to the standard deviation of the noise factors.

The generated instance response y is assumed to be a third order polynomial in the independent variables x_i's. The coefficients $_i$'s are the main effects. The coefficients $_{ij}$'s model two-way interactions including control by control, control by noise and noise by noise interactions. The coefficients $_{ijk}$'s model three-way interactions including control-by-control-by-control, control by control by noise, control by noise by noise and noise by noise by noise. The model originally proposed by Chipman, Wu and Hamada (1997) did not include three-way interactions. Li and Frey (2005) extended the model to included three-way interactions.

The values of the polynomial coefficients β's are determined by a random process that models the properties of effect sparsity, hierarchy, and inheritance. Equation 2.5 determines the probability density function for the first order coefficients. Factors can be

either "active" or "inactive" depending on the value (0 or 1 respectively) of their corresponding parameters $_i$'s. The parameter strength of active effects is assumed to be c times that of inactive effects. Equations 2.6 and 2.7 determine the probability density function for second order and third order coefficients respectively. In equations 2.6 and 2.7 the hierarchy principle is reflected in the fact that second order effects are only s_1 times as strong (on average) as first order effects ($s_1 \leq 1$) and third order effects are only s_2 times as strong as first order effects.

Equation 2.8 reflects sparsity of effects principle. There is a probability p of any main effect being active. Equation 2.9 and 2.10 enforce inheritance. The likelihood of any second order effect being active is low if no participating factor has an active main effect and is highest if all participating factors have active main effects. Thus generally one sets $p_{11} > p_{01} > p_{00}$ and so on.

We classified systems and response surface instances based on hierarchical ordering principle. The classes are:

- **Strong hierarchy** systems are ones which have only main effects and two-factor interactions active. Some small three-factor interactions might be present in such systems but they are not active.
- **Weak hierarchy** systems are ones which also have active three-factor interactions.

To generate a response surface instance, first the values of the probabilities of given factor effects being active (next section) are determined. Using them in equations 2.8 to 2.10, active effects for a given response surface instance are determined. Once active effects are known equations 2.5 to 2.7 are used to find the values of β's. Equations 2.2 to 2.4 are used to find the values of control factor, noise factors and experimental error for the model. To the find the instance's response, values of x_i's, β's and are substituted in equation 2.1.

2.3 Selecting parameters for Hierarchical Probability Model

Hierarchical Probability Model has several real valued parameters which have significant effect on the inferences drawn from its use. To provide a balanced view Frey and Li (2004) in tables 2.1 and 2.2 share six different settings of parameter settings.

	c	s_1	s_2	w_1	w_2
Basic WH	10	1	1	1	1
Basic low w	10	1	1	0.1	0.1
Basic 2nd order	10	1	0	1	1
Fitted WH	15	1/3	2/3	1	1
Fitted low w	15	1/3	2/3	0.1	0.1
Fitted 2nd order	15	1/3	0	1	1

Table 2.1: Parameters for Hierarchical Probability Model

	p	p_{11}	p_{01}	p_{00}	p_{111}	p_{011}	p_{001}	p_{000}
Basic WH	0.25	0.25	0.1	0	0.25	0.1	0	0
Basic low w	0.25	0.25	0.1	0	0.25	0.1	0	0
Basic 2nd order	0.25	0.25	0.1	0	N/A	N/A	N/A	N/A
Fitted WH	0.43	0.31	0.04	0	0.17	0.08	0.02	0
Fitted low w	0.43	0.31	0.04	0	0.17	0.08	0.02	0
Fitted 2nd order	0.43	0.31	0.04	0	N/A	N/A	N/A	N/A

Table 2.2: Additional parameters for Hierarchical Probability Model

The basic weak heredity model (basic WH) is based on the parameters used in Bayesian model selection, Chipman, Wu and Hamada (1997). Two variants were developed from this basic model. The low w variant accounts for the fact that control factors are generally explored over a wider range than noise factors. The 2nd order variant zeros out the coefficients of all the three-factor interactions. The fitted weak heredity model (fitted WH) was developed by Frey and Li (2004) based on experimental data.

Through out this thesis several different variants of these model parameters will be used. We will also use other parameter values, different from the ones as given over here to explore some effects in greater details. This would be done specifically to study the effectiveness of Compound Noise strategy and Take-The-Best-Few strategy on systems showing *effect sparsity*.

2.4 Variants of Hierarchical Probability Model

To study the impact of neglecting correlation among noise factors on Robust Design Studies, some more variants of Hierarchical Probability Model were developed. In particular equation 2.2 was modified as:

$$x_i \sim N(0, \mathbf{K}) \; i \in 1 \ldots m \qquad \textbf{(2. 11)}$$

Equation 2.11 shows that the first set of m input parameters to Hierarchical Probability Model ($x_1, x_2, \ldots, x_m$) are regarded as noise factors and are assumed to be normally distributed with variance-covariance **K** among noise factors.

Multiple variants of the hierarchical probability model were formed by selecting different sets of model parameters as described in Table 2.3. As the column headings of Table 2.3 indicate, a key difference between the variants is the assumption concerning effect hierarchy. A Strong Hierarchy Model assumes that the only active effects in the system are main effects and two-factor interactions although small three factor interactions are present as can be seen in Equation 2.7. A Weak Hierarchy Model includes a possibility for active three-factor interactions. The values for parameters in Table 2.3 such as p_{11}=0.25 and p_{01}=0.1 are based on the Weak Heredity model proposed by Chipman, Hamada and Wu (1997). In fact, the Strong Hierarchy model is precisely the Weak Heredity model published in that paper and used for analyzing data from experiments with complex aliasing patterns. The Weak Hierarchy model proposed here is an extension of that model to include higher order effects and therefore relies less on the assumption of hierarchy.

Additional model variants are based on the options in the last three rows of Table 2.3. The on-diagonal elements of the covariance matrix were varied among two levels. The

covariance matrix was also composed by three different methods inducing different degrees of correlation. These resulted in off-diagonal elements of the covariance matrix with different average magnitudes. Given the two model options related to the columns of Table 2.3 and the additional combinations of options due to the alternatives in the last three rows, there are 24 different model variants in all.

parameters	**Strong Hierarchy Model (active main effects and two-factor interactions)**	**Weak Hierarchy Model (active three-factor interactions also included)**
m	5	5
n	12	12
c	10	10
p	0.25	0.25
p_{11}	0.25	0.25
p_{01}	0.1	0.1
p_{00}	0.0	0.0
p_{111}	0.0	0.25
p_{011}	0.0	0.1
p_{001}	0.0	0.0
p_{000}	0.0	0.0
σ_ε	1 or 10	1 or 10
$\mathbf{K}_{ii}$	1.0 or 1.75	1.0 or 1.75
$\mu(\mathbf{K}_{ij} \mid), i \neq j$	0.01, 0.26, or 0.47	0.01, 0.26, or 0.47

Table 2.3: Parameters for Variants of Hierarchical Probability Model

K is the variance-covariance matrix for the real noise factors. The modeled noise, in response surface instance is assumed to have a covariance of an identity matrix. Thus the more different **K** is from an identity matrix, the more the noise strategy varies from a faithful representation of the noises the system will experience in the field.

The on-diagonal elements of the matrix, $\mathbf{K}_{ii}$, are the variance due to each noise factor x_i. The size of these on-diagonal elements is an indication of the amplitude of the real noise factors relative to the modeled noise factors. Two options within the model are defined: one in which the real noise has the same variance as the modeled, and one in which the real noise has higher variance than the modeled.

The off-diagonal elements of the matrix, $\mathbf{K}_{ij}$, are the covariance among noise factors x_i and x_j. Three options within the model are defined: one with almost no correlation (with the average absolute value of the correlation coefficients being 0.01), one with relatively mild correlation (with the average absolute value of the correlation coefficients being 0.26) and one with relatively strong correlation (with the average absolute value of the correlation coefficients being 0.47). The matrix $\mathbf{K}$ was formed so as to ensure the resulting matrix was positive semi-definite while also having the desired variance and the desired degree of correlation.

2.5 Chapter Summary

In this chapter the formulation of Hierarchical Probability Model was discussed. This will form the basis to compare different robust design methods statistically. First the regularities exhibited by engineering systems were discussed. Next those regularities were put in a mathematical format. The mathematical formulation would be used to generate response surface instances to analyze different robust design methods. We also discussed about selecting various parameters for Hierarchical Probability Model. We can

have many variants of Hierarchical Probability Model. We discussed some of these variants.

In the next chapters some robust design methods will be discussed, which focus on reducing number of experiments done on systems and reducing amount of information required about system and still improve robustness of the system. We will use Hierarchical Probability Model as one of the basis to analyze these robust design methods.

Chapter 3: Compound Noise: Evaluation as a Robust Design Method

3.1 Introduction and Background

In Robust Parameter Design methodology, the effect of noise (variation) is reduced by exploiting control-by-noise interactions. These control-by-noise interactions can be captured by using crossed-array approach. The control factor setting that minimizes the sensitivity of the response to noise factors is called the optimal control factor setting or the most robust setting for the system. A crossed-array approach is a combination of two orthogonal arrays, one of control factors and other of noise factors. Exploiting control-by-noise interactions is just the beginning of reducing sensitivity of the response. For systems that have active three-factor interactions, control-by-control-by-noise and control-by-noise-by-noise interactions can also be utilized to reduce the sensitivity of system's response. But as the complexity of the system increases, use of full factorial control and noise factor arrays becomes prohibitively expensive. As an attempt to reduce the run size of this crossed-array approach, Taguchi (1987) proposed a compound noise factor technique. A compound noise factor is typically formed by combining all the noise factors of a system into a single factor, which is used instead of noise array.

When we know which noise factor levels cause the output to become large or small they are compounded so as to obtain one factor with two or three levels. On doing this, our

noise factors become a single compounded factor, no matter how many factors are involved. Taguchi (1987) and Phadke (1989) outlined conditions on the use and formulation of compound noise. Noise factors can be combined into a single compound factor based on their directionality on the response. Directionality of effects of noise factors on the response can be found by running small number of experiments. Phadke (1989) showed how to construct a compound noise factor using the results from the Operational Amplifier and Temperature Control Circuit.

Du, et al. (2003, 2004) used percentile performance difference of the system to construct compound noise. This method is applicable for systems which show unimodal response characteristics. The compound noise formed this way also carries information about sensitivity of system's response to noise variables. In this thesis compound noise will be formed by the algorithm given by Phadke (1989). We will determine directionality of the effects of noise factors on the system's response and will combine noise factors based on their directionality.

Hou (2002) studied the conditions that will make compound noise yield robust setting for systems. Hou said "*extreme settings should exist for compound noise to work*". We will later find that compound noise can be effective even when extreme settings do not exist. The conditions mentioned in "*Compound Noise Factor Theory*" turn out to be the sufficient conditions. In later sections the analysis will be extended to determine conditions under which compound noise will predict a robust setting. Hou's formulation was limited to systems which had active effects up to two-factor interactions. We will

extend the formulation to systems which can have active effects up to three-factor interactions.

Compound Noise can be considered an extension of supersaturated designs (SSD). This concept initially originated with a paper by Satterthwaite (1959). SSDs were assumed to offer a potentially useful way to investigate many factors with few experiments. In some SSDs the number of factors being investigated may exceed the number of experiments by a large factor. Holcomb, et al. (2002) discusses the construction and evaluation of SSDs. Holcomb, et al. (2003) outlines the analysis of SSDs. Compound Noise is an unbalanced SSD. Allen and Bernshteyn (2003) discuss the advantages of unbalanced SSDs in terms of performance and affordability. Heyden, et al. (2000) argues SSDs can be used to estimate variance of response, which can be used as a measure of robustness rather than using it to find main effects. SSDs "*do not allow estimation of the effects of the individual factors because of confounding between the main effects*". But "*estimation of the separate factor effects is not necessarily required*" in improving robustness. Using compound noise as a robust design method we try to estimate robustness of the system at a given control factor setting. The setting which improves this estimate is taken as the predicted robust setting.

The main aim of this chapter is to explore the effectiveness of compound noise as a robust design method. We will first look at the effectiveness of compound noise strategy on response surface instances generated using Hierarchical Probability Model, Li and Frey (2005). Two different kinds of response surface instances were generated. One with

only main effects and two factor interactions also called as *strong hierarchy* instances and other with main effects, two factor interactions and three factor interactions, also called as *weak hierarchy* instances.

The conclusions from compound noise studies on response surface instances from Hierarchical Probability Model were then verified by testing compound noise on six case studies from different engineering domains. The thesis also provides theoretical justification for the effectiveness of a compound noise strategy. This thesis also gives an alternative strategy to formulate a compound noise, distinctly different from Taguchi's formulation of compound noise. This new compound noise strategy requires no knowledge about noise factors effect for its formulation. The main take away of this chapter is that, Compound Noise as a Robust Design Method was very effective on the response surface instances and case studies for which only few effects accounted for significant impact on response.

The chapter is organized as: first we will describe setting up of compound noise strategy on response surface instances, measures we studied, results from compound noise studies followed by conclusions from Hierarchical Probability Model. Then compound noise strategy for six case studies from different engineering domains will be formulated. We will also study the conditions which lead to effectiveness of Compound Noise strategy. In the end conclusions and summary of the chapter will be given.

3.2 Setting up of Compound Noise study

The objective of this chapter is to see the effect of compounding noise on the prediction of optimal setting of a response surface instance generated by hierarchical probability model parameters and verify the results by case studies from various engineering domains. Here two kinds of compounding schemes were studied: **simple compounding** of noise factors and **extreme compounding** of noise factors. In the field when the direction of a system's response due to changing each of the noise factors is known, extreme compound noise can be constructed. This formulation of compound noise was originally proposed by Taguchi (1987) and Phadke (1989). We make a two-level extreme compound noise factor by combining noise factor levels in such a way that, at the lower setting of extreme compound noise all the noise factor settings give minimum response and vice versa. When there is no idea about the direction in which noise factors affect a system's response, simple compound noise is constructed. In this case noise factor levels are simply combined to form two-level simple compound noise. For example, we can select the setting of noise factors randomly to form lower-level compound noise.

To construct extreme compound noise strategy, some knowledge about effect of noise factors on system's response is needed. Hence it is in some sense more expensive to construct for a given system. But no such information is required about effect of noise factors while formulating simple compound noise strategy. Simple compound noise strategy is less resource intensive as compared to extreme compound noise strategy. We will apply both of these compound noise strategies to strong and weak hierarchy response surface instances and analyze robustness gain. We will explore the reasons behind

success or failure of compound noise application for response surface instances. The questions we want to address in this chapter are:

- Why is compound noise effective in achieving robust setting in certain cases, but ineffective in other cases?
- How can we measure the effectiveness of a compound noise strategy?
- Do we need to know the directionality of noise factors to use compound noise? (Simple Compound Noise strategy vs. Extreme Compound Noise strategy)

3.2.1 Generating Response Surface instances

To study the effectiveness of compound noise strategy we will generate instances of response surfaces using Hierarchical Probability Model. The instances of response surfaces studied had 7 control factors and 5 noise factors. These response surfaces were generated according to a relaxed weak heredity model. The full third-order response surface equation is given by

$$y(x_1, x_2, \ldots, x_{12}) = \sum_{i=1}^{12} \beta_i x_i + \sum_{i=1}^{12} \sum_{\substack{j=1 \\ j>i}}^{12} \beta_{ij} x_i x_j + \sum_{i=1}^{12} \sum_{\substack{j=1 \\ j>i}}^{12} \sum_{\substack{k=1 \\ k>j}}^{12} \beta_{ijk} x_i x_j x_k + \varepsilon \qquad \textbf{(3.1)}$$

where x_1-x_5 represent normally distributed random noise variables, x_6- x_{12} represent control factors, $\beta' s$ are factor coefficients and ε is experimental error.

In our study, we generated both strong and weak hierarchy response surface instances. The noises factors were assumed to be uncorrelated. Full factorial array was used for 7 control factors. Table 3.1 and table 3.2 give hierarchy probability model parameter values used to generate strong and weak hierarchy response surface instances respectively.

parameters	*values*
c	15
s_1	0.33
s_2	0.67
w_2	1
p	0.43
p_{11}	0.31
p_{01}	0.04
p_{00}	0

Table 3.1: Hierarchical Probability Model Parameters used for Strong Hierarchy Response Surface instances

parameters	*values*
c	15
s_1	0.33
s_2	0.67
w_2	1
p	0.43
p_{11}	0.31
p_{01}	0.04
p_{00}	0.0
p_{111}	0.17
p_{011}	0.08
p_{001}	0.02
p_{000}	0.0

Table 3.2: Hierarchical Probability Model Parameters used for Weak Hierarchy Response Surface instances

To gaze the impact of *effect sparsity* on the effectiveness of compound noise strategy, we changed parameter values for both strong and weak hierarchy response surfaces. The changed parameters are given in tables 3.3 and 3.4 for strong and weak hierarchy response surface instances respectively.

parameters	***values***
c	15
s_1	0.33
s_2	0.67
w_2	1
p	0.3
p_{11}	0.15
p_{01}	0.04
p_{00}	0

Table 3.3: Hierarchical Probability Model Parameters used for Strong Hierarchy Response Surface instances, reflecting *effect sparsity*

parameters	*values*
c	15
s_1	0.33
s_2	0.25
w_2	1
p	0.3
p_{11}	0.15
p_{01}	0.04
p_{00}	0.0
p_{111}	0.03
p_{011}	0.03
p_{001}	0.02
p_{000}	0.0

Table 3.4: Hierarchical Probability Model Parameters used for Weak Hierarchy Response Surface instances, reflecting *effect sparsity*

For noise factors we defined 3 different noise strategies. First was a 2_V^{5-1} noise array. This noise strategy is very close to full factorial noise factor array. Hence it will give almost perfect results for generated response surface instances. This will form a basis of comparison for compound noise strategies. Second was simple compound noise, in which noise factors settings were selected randomly to form two-level simple compound noise. Third was extreme compound noise, in which noise factors were compounded based on the sign of their effect coefficient in the response surface instance. The way simple and extreme compounded noises were formed for the 5 noise factors, is shown below, equations 3.2 and 3.3.

Simple Compounded Noise Factor levels

$$\begin{bmatrix} - \\ + \end{bmatrix} = \overbrace{\begin{bmatrix} random(sign(\beta_1)) & \cdots & \cdots & \cdots & random(sign(\beta_5)) \\ -\Downarrow & \cdots & \cdots & \cdots & -\Downarrow \end{bmatrix}}^{N_1 \; N_2 \; N_3 \; N_4 \; N_5} \tag{3.2}$$

Extreme Compounded Noise Factor levels

$$\begin{bmatrix} - \\ + \end{bmatrix} = \overbrace{\begin{bmatrix} -sign(\beta_1) & -sign(\beta_2) & -sign(\beta_3) & -sign(\beta_4) & -sign(\beta_5) \\ sign(\beta_1) & sign(\beta_2) & sign(\beta_3) & sign(\beta_4) & sign(\beta_5) \end{bmatrix}}^{N_1 \; N_2 \; N_3 \; N_4 \; N_5} \tag{3.3}$$

where $\beta's$ are the coefficient of noise factors for a given instance of response surface as predicted by hierarchy probability model parameters.

For each of the three noise strategies, we generated 200 response surface instances. For each instance of generated response surface, we used Monte Carlo on noise factors to find response variance at each level of full-factorial control factor array. The setting of control factors giving minimum variance of response was the optimal setting for that instance of response surface.

We analyzed each response surface instance by running a designed experiment using 3 different robust design crossed arrays: $2^7 \times 2_V^{5-1}$ (Noise Strategy 1), $2^7 \times$ Simple Compound Noise strategy and $2^7 \times$ Extreme Compound Noise strategy. For each crossed array experiment we predicted optimal setting of control factors of response surface, by finding minimum response variance for that experiment. This predicted optimal setting of

response surface under each crossed array experiment was then compared with optimal setting we got by using Monte Carlo on noise factors. In the next section we will present this algorithm in a diagrammatic manner.

3.2.2 Algorithm to study Compound Noise

The algorithm used to study noise compounding is shown in figure 3.1. The same algorithm was used to study both strong and weak hierarchy response surface instances. For each generated response surface instance, we studied three different robust design methods as outlined in previous section. Left hand side of the algorithm finds optimal setting of response surface instance, using Monte Carlo on noise factors. Right hand side of the algorithm finds optimal setting of response surface instance as predicted by a chosen robust design method ($2^7 \times 2_V^{5-1}$ (Noise Strategy 1), $2^7 \times$ Simple Compound Noise strategy or $2^7 \times$ Extreme Compound Noise strategy). Various measures are finally compared to compute the effectiveness of a chosen robust design strategy.

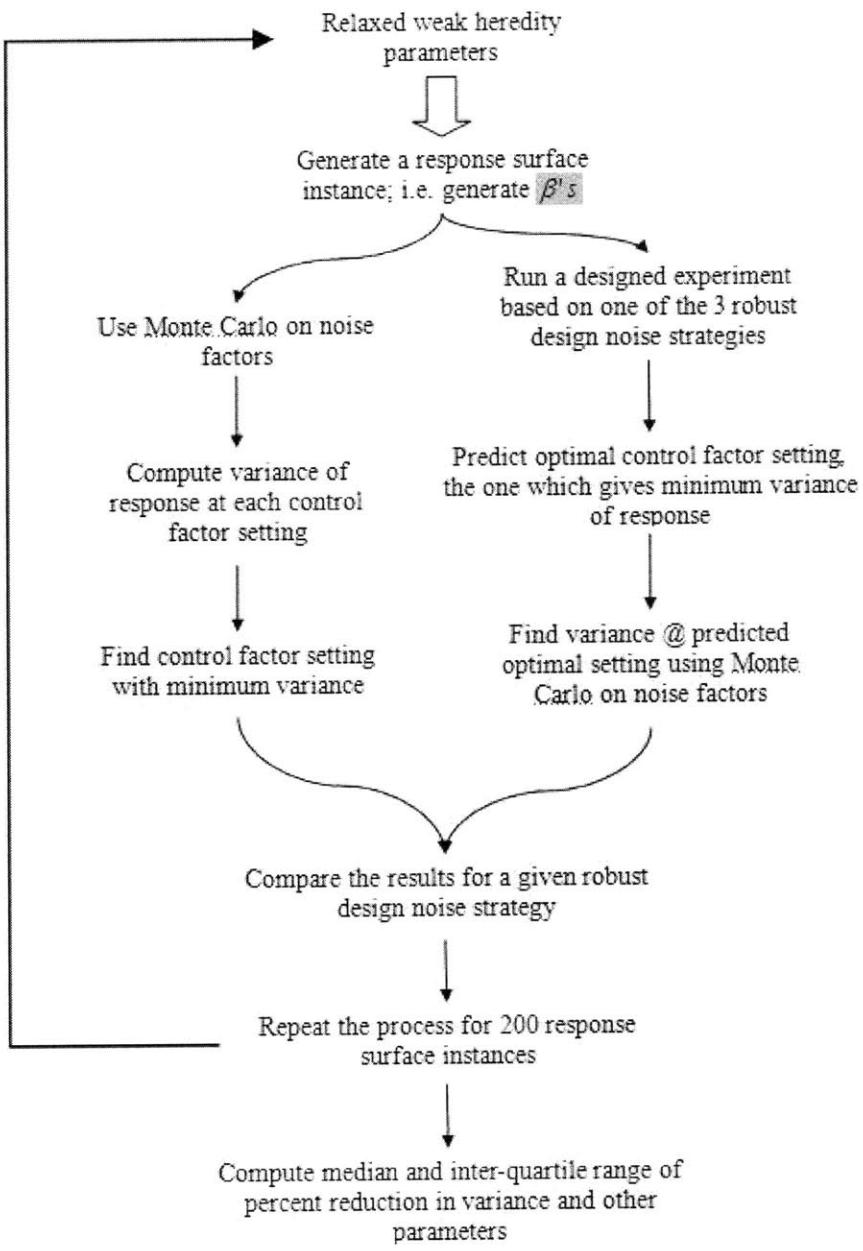


Figure 3.1: Algorithm used to evaluate Compound Noise Strategies

3.2.3 Measures studied

There were number of measures that were derived from the above study. The first was the percentage number of times a given noise strategy could predict the optimal control factor setting, as given by running Monte Carlo simulation at each control factor setting for each generated instance of response surface.

For the second measure we found the predicted optimal settings of the 7 individual control factors as given by a noise strategy. We compared the predicted optimal setting of each control factor with the optimal control factor setting as given by Monte Carlo simulation. We then found number of control factors out the 7, which had same setting as given by noise strategy and Monte Carlo simulation. The second measure was the mean of this number for 200 instances that were generated.

For the third measure, we first found the predicted optimal control factor setting by using a given noise strategy. Then we ran Monte Carlo simulation on noise factors at that predicted optimal control factor setting and calculated the standard deviation labeled $_{strategy}$. The minimum standard deviation found by running Monte Carlo simulation at each control factor setting is called σ_{opt}. The third measure was the ratio of $\sigma_{strategy}$ to σ_{opt} for each generated instance of response surface. We studied the median and inter-quartile range of this measure for 200 instances that were generated.

For the fourth measure, we calculated the mean of the standard deviations, as given by Monte Carlo at each control factor setting. We called this mean as σ_{base}. This σ_{base} was

our reference. (σ_{base} - $_{opt}$) is the maximum reduction that is possible in standard deviation for a given instance of response surface. But by running a given noise strategy the realized reduction would only be (σ_{base} - $_{strategy}$). The fourth meaure was the ratio of realized reduction to the maximum reduction possible, i.e. ratio of (σ_{base} - $_{strategy}$) to (σ_{base} - $_{opt}$) for each generated instance. And we studied the median and inter-quartile range of this measure for 200instances that were generated. Figure 3.2 shows diagrammatic representation of positive improvement ratio for an instance of generated response surface.

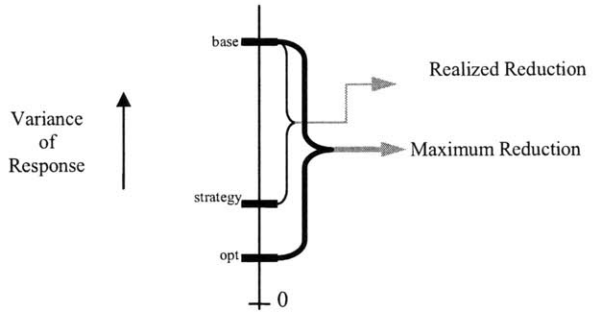


Figure 3.2: Improvement Ratio: Ratio of Realized Reduction to Maximum Reduction possible

3.2.4 Results from Compound Noise studies

The results from Compound Noise studies on Strong and Weak Hierarchy response surface instances are presented from tables 3.5 to 3.10.

Percent Matching of ALL Control Factors with their Robust Setting				
	Strong Hierarchy response surfaces (*sparse effects*)	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces (*sparse effects*)	Weak Hierarchy response surfaces
Noise Strategy 1	72%	70%	63%	61%
Simple Compounding	8%	6%	3%	3%
Extreme Compounding	10%	8%	4%	4%

Table 3.5: Measure 1

Percent Matching of Control Factors with their Robust Setting				
	Strong Hierarchy response surfaces (*sparse effects*)	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces (*sparse effects*)	Weak Hierarchy response surfaces
Noise Strategy 1	93%	90%	82%	80%
Simple Compounding	55%	50%	53%	49%
Extreme Compounding	73%	61%	54%	49%

Table 3.6: Measure 2

Median $\left(\frac{\sigma_{strategy}}{\sigma_{opt}}\right)$				
	Strong Hierarchy response surfaces (*sparse effects*)	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces (*sparse effects*)	Weak Hierarchy response surfaces
Noise Strategy 1	1.00	1.00	1.00	1.00
Simple Compounding	1.12	1.25	1.49	1.56
Extreme Compounding	1.08	1.12	1.36	1.41

Table 3.7: Measure 3

25^{th} and 75^{th} percentile $\left(\frac{\sigma_{strategy}}{\sigma_{opt}}\right)$				
	Strong Hierarchy response surfaces (*sparse effects*)	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces (*sparse effects*)	Weak Hierarchy response surfaces
Noise Strategy 1	1.00-1.00	1.00-1.00	1.00-1.03	1.00-1.03
Simple Compounding	1.04-1.35	1.12-1.51	1.25-1.79	1.27-1.73
Extreme Compounding	1.02-1.16	1.13-1.62	1.15-1.64	1.15-1.73

Table 3.8: Measure 3

Median $\left(\frac{\sigma_{base} - \sigma_{strategy}}{\sigma_{base} - \sigma_{opt}}\right)$				
	Strong Hierarchy response surfaces (*sparse effects*)	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces (*sparse effects*)	Weak Hierarchy response surfaces
Noise Strategy 1	1.00	1.00	1.00	1.00
Simple Compounding	0.68	0.52	0.33	0.29
Extreme Compounding	0.82	0.69	0.54	0.43

Table 3.9: Measure 4

25^{th} and 75^{th} percentile $\left(\frac{\sigma_{base} - \sigma_{strategy}}{\sigma_{base} - \sigma_{opt}}\right)$				
	Strong Hierarchy response surfaces (*sparse effects*)	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces (*sparse effects*)	Weak Hierarchy response surfaces
Noise Strategy 1	0.99-1.00	0.98-1.00	0.96-1.00	0.95-1.00
Simple Compounding	0.24-0.88	0.14-0.75	0.16-0.64	0.11-0.63
Extreme Compounding	0.53-0.94	0.31-0.87	0.23-0.80	0.20-0.74

Table 3.10: Measure 4

As we can see from above tables that as the effects become less sparse (i.e. more dense) improvement in robustness that can be achieved for both Strong and Weak Hierarchy response surface instances decreases. To explore this effect further, we varied *effect density* over a wide range and plotted median improvement ratio for both Strong and Weak Hierarchy response surface instances. Figures 3.3 to 3.5 present the results from the above study.

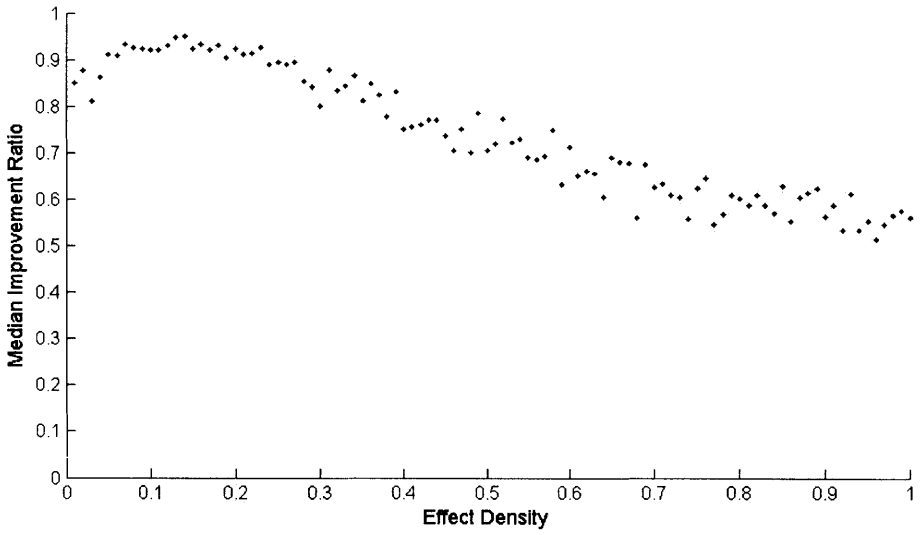


Figure 3.3: Median Improvement Ratio verses Effect Density for Strong Hierarchy Response Surface Instances

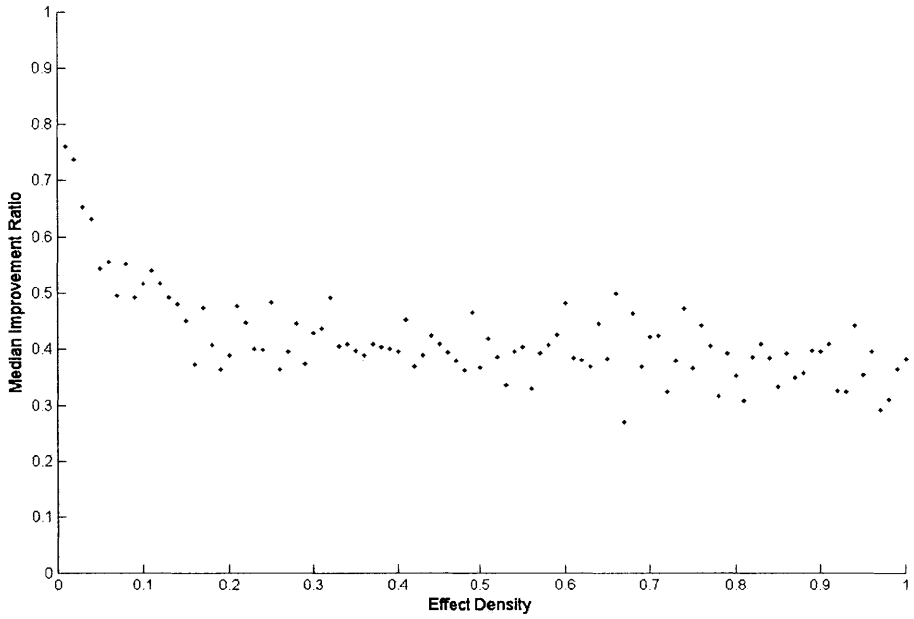


Figure 3.4: Median Improvement Ratio verses Effect Density for Weak Hierarchy Response Surface Instances

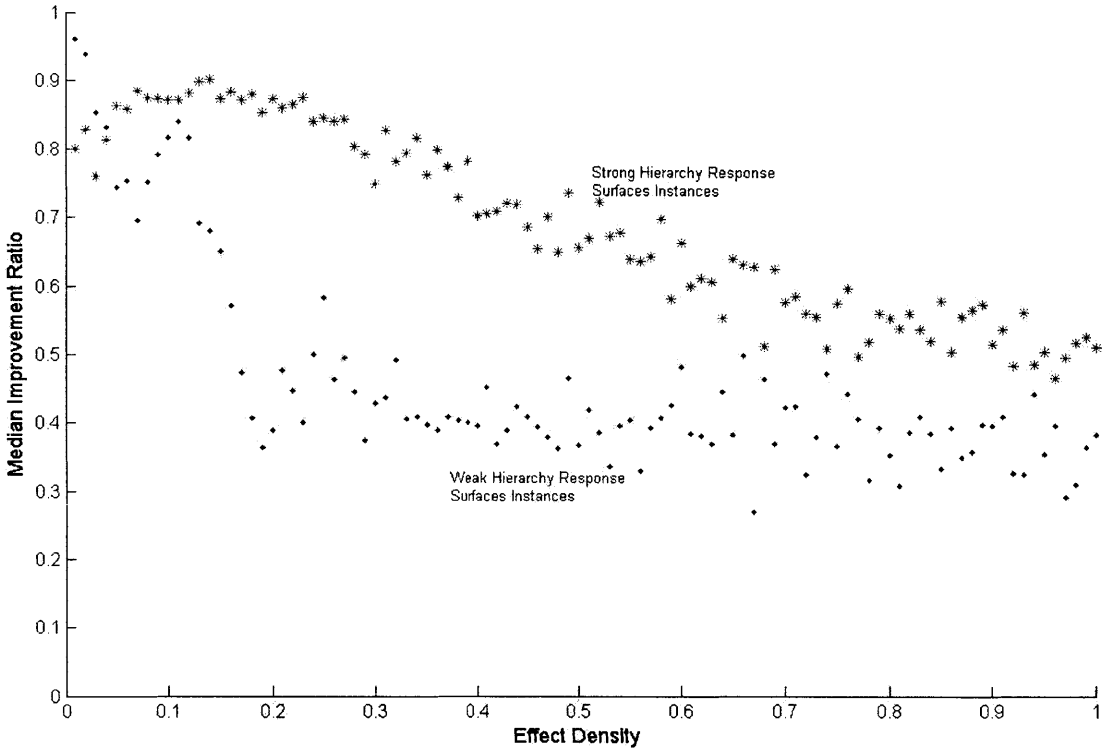


Figure 3.5: Median Improvement Ratio verses Effect Density for both Strong and Weak Hierarchy Response Surface Instances

3.2.5 Conclusions from Probability Model

The use of compound noise as robust design method leads to reduction in experimental effort. Up till now we have tried to gage the effectiveness of compound noise as a robust design method for response surface instances generated using Strong and Weak Hierarchical Probability Model. As we can see from above tables and figures that compound noise strategy is very effective for instances which show *effect sparsity*. We used two formulations of compound noise strategies. One of them was adopted from

Taguchi (1987) and Phadke (1989), *extreme compound noise*. For such a formulation we need to know the directionality of noise factors on system's response. The other formulation was *simple compound noise*. For simple compound noise strategy we randomly combine the settings of noise factors, and we do not need to know the directionality of noise factors. We also found that even such a formulation of compound noise strategy was very effective in obtaining high robustness gains, for response surface instances showing *effect sparsity*. The reason why such a formulation of compound noise is very effective is that active effects in response surface instances are sparse.

As active effects present in response surface instance become more densely populated, the percentage improvement that can be achieved using compound noise strategies decreases. This happens for response surface instances generated via both Strong and Weak Hierarchical Probability Model.

In the next section we will analyze compound noise strategies for six case studies from various engineering domains. We will explore the reasons for the effectiveness or ineffectiveness of compound noise strategies.

3.3 Case Studies

In the previous section we have seen that both extreme and simple formulations of compound noise strategy work good in improving robustness gain for instances of response surfaces that show *effect sparsity*. To provide a further check on these results, we identified and implemented six system simulations, three of them followed strong

hierarchy and three followed weak hierarchy. Using these six case study simulations from various engineering domains, we performed robust design studies using various compound noise strategies as described in previous sections. The three case studies that exhibited strong hierarchy are Operational Amplifier (Op Amp), Phadke (1989), Passive Neuron Model, Tawfik and Durand (1994), and Journal Bearing: Half Sommerfeld Solution, Hamrock, et al. (2004). The three case studies that exhibited weak hierarchy are Continuous-Stirred Tank Reactor (CSTR), Kalagnanam and Diwekar (1997), Temperature Control Circuit, Phadke (1989) and Slider Crank, Gao, et al. (1998). For each case study we ran full factorial control and noise factor crossed array experiments. These runs were used to determine the effect coefficients for main effects, two-factor interactions and three-factor interactions. Lenth's Method, Lenth (1989), was employed to determine the active effects. The full factorial results were also used to determine the most robust setting for all six systems.

3.3.1 Lenth Plots for Case Studies

For each of the six case studies we ran full factorial control and noise factor crossed array. The results from these robust design experiments were used to determine the effect coefficients for main effects, two-factor interactions and three-factor interactions. Figures 3.6 to 3.11 give Lenth plots for all six case studies, to determine active main effects present in them.

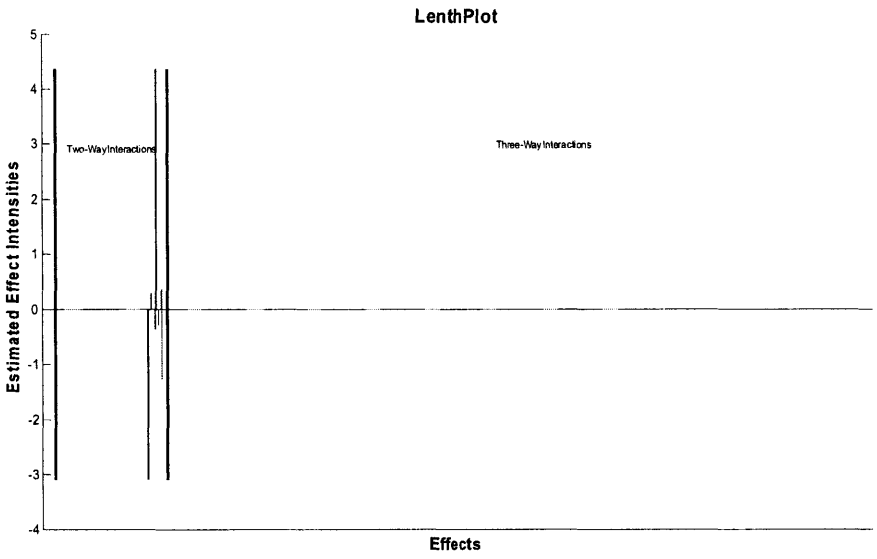


Figure 3.6: Lenth Plot of Effect Coefficients for Op Amp, Phadke (1989)

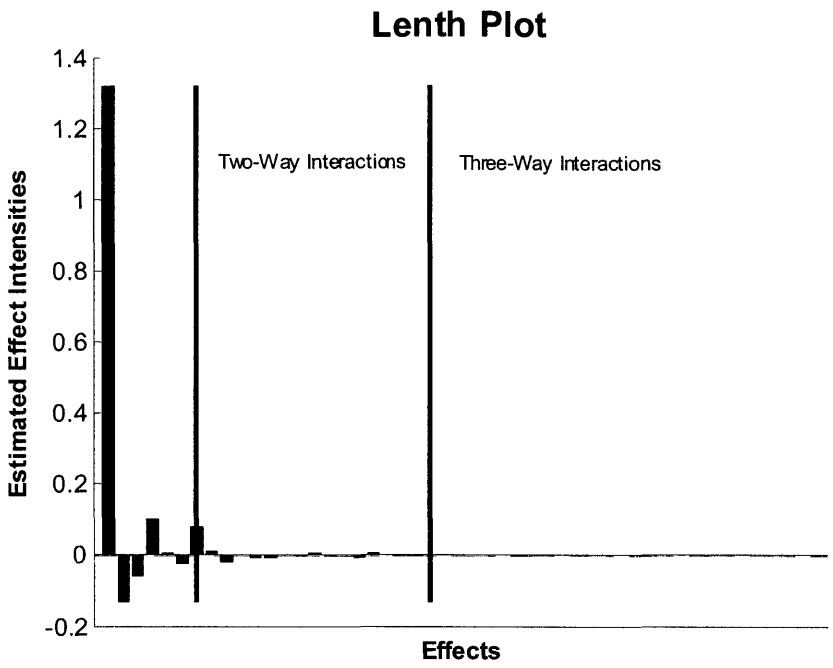


Figure 3.7: Lenth Plot of Effect Coefficients for Passive Neuron Model, Tawfik and Durand (1994)

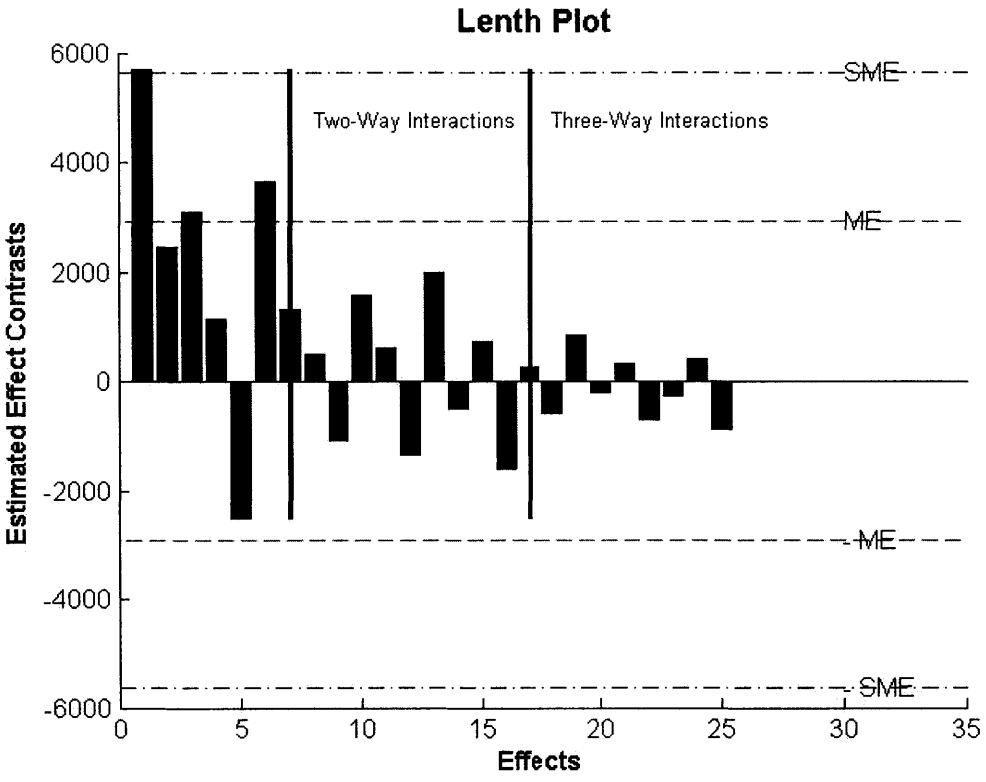


Figure 3.8: Lenth Plot of Effect Coefficients for Journal Bearing: Half Sommerfeld Solution, Hamrock, et al. (2004)

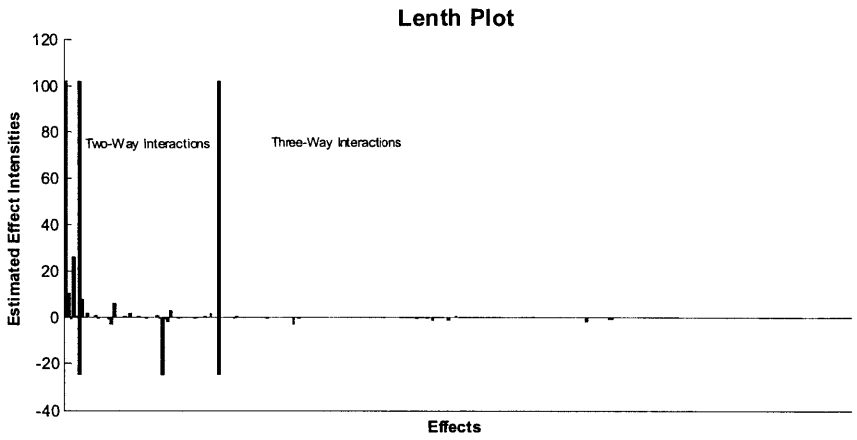


Figure 3.9: Lenth Plot of Effect Coefficients for CSTR, Kalagnanam and Diwekar (1997)

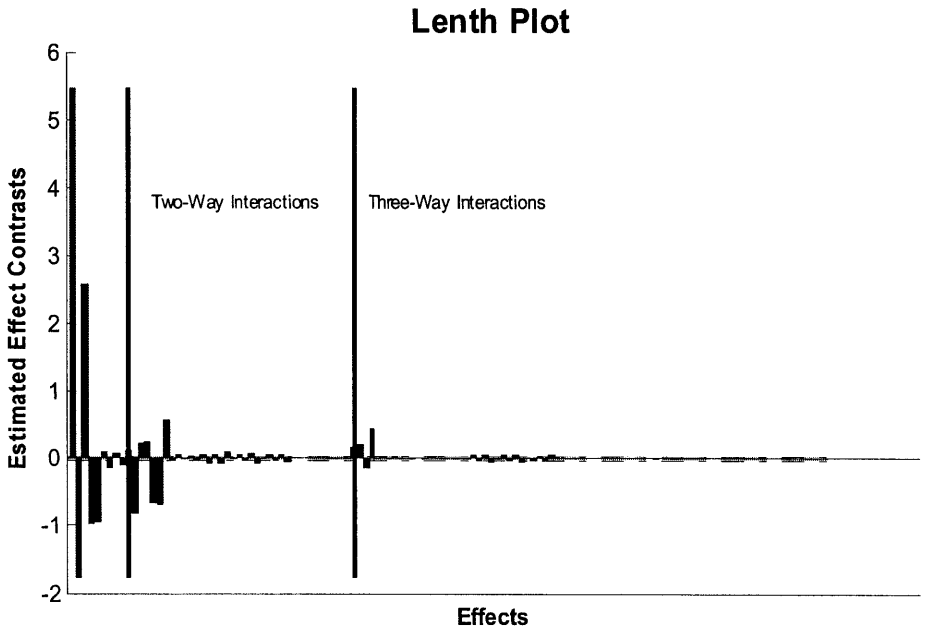


Figure 3.10: Lenth Plot of Effect Coefficients for Temperature Control Circuit, Phadke (1989)

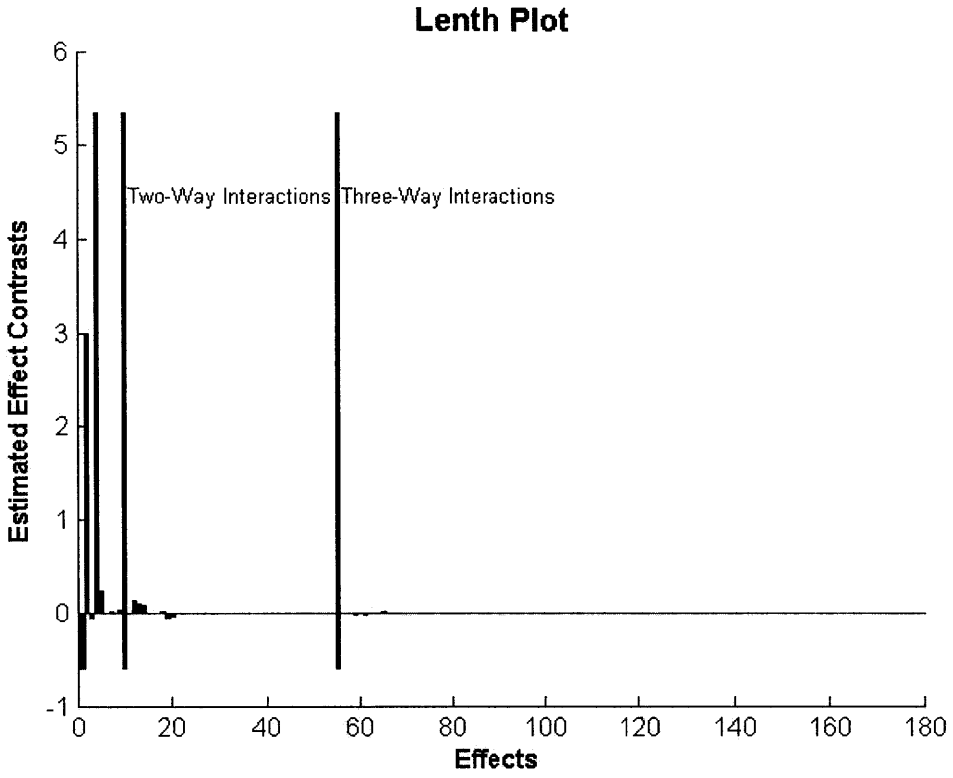


Figure 3.11: Lenth Plot of Effect Coefficients for Slider Crank, Gao, et al. (1998)

3.3.2 Results from Compound Noise Strategy on Case Studies

In order to formulate extreme compound noise for robust design experiments, we found the directionality of noise factors on the system response for all six systems. For each system we ran a full factorial design in control factors crossed with two-level compound noise. Pure experimental error was introduced in the system's response. The error was of the order of active main effects. From Lenth's Method, we can estimate the average magnitude of active main effects. Pure experimental error introduced into the system's response was normally distributed with zero mean and standard deviation as the average

magnitude of active main effects. Extreme compound noise strategy performed extremely well for the PNM, Op Amp, Journal Bearing and Slider Crank. Table 3.11 shows results from robust design experiments.

System	Hierarchy	Measure of Sparsity of Effects	Existence of Extreme Settings, Hou (2002)	Number of Replications	Percent Matching of **All** Control Factors with their Robust Setting	Percent Matching of Control Factors with their Robust Setting	Improvement Ratio
Op Amp	Strong	0.105	Existed	100	96%	99.2%	99.2%
PNM	Strong	0.132	Not-existed	100	98%	99%	99.5%
Journal Bearing	Strong	0.28	Existed	100	97.8%	98.8%	98.84%
CSTR	Weak	0.647	Not-existed	100	10%	55.17%	58.4%
Temp. Control Circuit	Weak	0.725	Not-existed	100	5%	29.25%	30.65%
Slider Crank	Weak	0.119	Existed	100	95%	98%	98.5%

Table 3.11: Results from Full Factorial Control Factor array and Two-Level Extreme Compound Noise

The third column of Table 3.11 shows the measure of sparsity of effects, which is the ratio of active effects in a system to total number of effects that can be present in the system. The sparsity of effects means that among several experimental effects examined in any experiment, a small fraction will usually prove to be significant, Box and Meyer (1986), Hamada and Wu (1992), Wu and Hamada (2000). The sixth column of Table 3.11 shows how well the predicted robust setting from compound noise experiments matches with the actual robust setting as found by carrying out full factorial control and noise array experiments. The seventh column of Table 3.11 shows the percent of control factors whose robust setting matched for full factorial and compound noise experiments.

At the predicted robust setting of control factors from compound noise experiments, we found corresponding variance of system's response from full factorial experiments. Full factorial experiments yield the optimal variance of the system's response and average variance of response at all control factor settings. With this knowledge, variance can be improved from the average value to the optimal value, which is the maximum possible improvement. For compound noise experiments variance of response can be improved from the average value to the variance at the predicted robust setting. The ratio of achieved improvement to maximum possible improvement is called improvement ratio. The average of this measure for all replicates is shown in the eighth column of Table 3.11.

From Table 3.11 we can see that extreme compound noise strategy works reasonably well for the Op Amp, PNM, Journal Bearing and Slider Crank, but not nearly as well for the CSTR, and Temperature Control Circuit.

3.4 Effectiveness of Compound Noise in Real scenarios

In real scenarios it is often not possible to run full factorial inner arrays like those reported on in Table 3.11. Therefore we also studied the use of fractional factorial arrays by running robust design experiments which were resolution III in both the control and noise factor array. We predicted robust settings from such experiments and compared the results to those from full factorial experiments in Table 3.12. We also ran robust design experiments which were resolution III in the control factor array combined with two-level extreme compound noise. Results for these experiments are summarized in Table 3.13.

Resolution III experiments formed the basis of comparison for the results we got from resolution III control factor array crossed with two-level extreme compound noise. Pure experimental error was introduced into the system's response, for all six case studies. The error introduced was of the order of active main effects. The control factors were randomly assigned to Resolution III inner array.

We did not study Passive Neuron Model and Journal Bearing systems because these systems had only two control factors and thus had no distinct resolution III control factor array.

System	Hierarchy	Measure of Sparsity of Effects	Number of Replications	Percent Matching of **All** Control Factors with their Robust Setting	Percent Matching of Control Factors with their Robust Setting	Improvement Ratio
Op Amp	Strong	0.105	100	98%	99.6%	99.91%
CSTR	Weak	0.647	100	NONE	76.5%	96.5%
Temp. Control Circuit	Weak	0.725	100	28%	55%	87%
Slider Crank	Weak	0.119	100	91%	95.5%	97.6%

Table 3.12: Results from Resolution III Control and Noise Factor Array

Table 3.12 indicates that the robust setting for the Op Amp and Slider Crank can be achieved with high probability when a resolution III noise factor array is used. For the CSTR and Temperature Control Circuit, the high mean improvement ratio means we will not attain the robust setting but will arrive near the optima. The high improvement ratio suggests that we will achieve most of the improvement possible in the variance of system's response.

System	Hierarchy	Measure of Sparsity of Effects	Number of Replications	Percent Matching of **All** Control Factors with their Robust Setting	Percent Matching of Control Factors with their Robust Setting	Improvement Ratio	Improvement Ratio (from Resolution III Noise Array)
Op Amp	Strong	0.105	100	86%	97.2%	99.37%	99.91%
CSTR	Weak	0.647	100	NONE	41.17%	39.8%	96.5%
Temp. Control Circuit	Weak	0.725	100	NONE	43.5%	25.2%	87%
Slider Crank	Weak	0.119	100	89%	94%	96.5%	97.6%

Table 3.13: Results from Resolution III Control Factor array and Two-Level Extreme Compound Noise

Table 3.13 indicates that even when using a resolution III control factor array, a compound noise strategy will predict the robust setting for the Op Amp and the Slider Crank with high probability. The compound noise strategy does not work too well with resolution III control factor array for the CSTR. The average improvement ratio possible is only 40%, as opposed to nearly 97% had the resolution III noise factor array been used. Nor does the compound noise strategy perform well with a resolution III control factor array for the Temperature Control Circuit. The improvement ratio drops from 87% for resolution III noise array to 25% for two-level extreme compound noise.

In the formulation of compound noise as suggested by Taguchi (1987) and Phadke (1989), we need to know the directionality of noise factors on system's response. We need to run some fractional factorial experiments on the system to gather information about directionality of noise factors. We tried a different formulation of compound noise. Noise factor settings can be picked randomly to form a two-level random compound

noise (simple compound noise strategy). We ran such a compound noise strategy with full factorial control factor array for each of the six case studies. Table 3.14 presents the average results from these experiments.

System	Hierarchy	Measure of Sparsity of Effects	Number of Replications	Percent Matching of **All** Control Factors with their Robust Setting	Percent Matching of Control Factors with their Robust Setting	Improvement Ratio	Improvement Ratio (from Taguchi's Compound Noise)
Op Amp	Strong	0.105	1000	30.8%	73%	80.2%	99.2%
PNM	Strong	0.132	2^4 =16	50%	68.75%	79.9%	99.5%
Journal Bearing	Strong	0.28	2^3 =8	100%	100%	100%	98.84%
CSTR	Weak	0.647	2^6 = 64	54.69%	70.3%	67.6%	58.4%
Temp. Control Circuit	Weak	0.725	2^5 = 32	68.75%	76.56%	74.75%	30.65%
Slider Crank	Weak	0.119	2^5 = 32	50%	70%	55.2%	98.5%

Table 3.14: Average results from Full Factorial Control Factor array and Two-Level Simple Compound Noise Strategy

The Op Amp has 21 noise factors, so there can be 2^{21} combinations of random compound noise. Instead we ran 1000 such combinations of random compound noise for the Op Amp. The Passive Neuron Model has 4 noise factors. Hence there are 2^4 combinations of random compound noise for the PNM. The Journal Bearing has three, the CSTR has six and the Temperature Control Circuit and Slider Crank have five noise factors each. From Table 3.14 we see that such a formulation of simple compound noise is very effective on an average. Except for the Slider Crank system, simple compound noise can achieve improvement ratios greater than 68% on an average, for all systems. For the Slider Crank system too, half of the replications gave robust setting of the system and half of them led

to poor settings. Hence had we used simple compound noise *with care* for the Slider Crank system we would have attained robust setting.

3.5 Conditions for Compound Noise to be completely effective

Hou (2002) gave conditions under which a compound noise strategy will predict the robust setting for a system. One of the conditions for compound noise to work is the existence of extreme settings. The extreme settings of compound noise are those which maximize and minimize the system's response to capture noise variations. But we found that in some systems (for example Passive Neuron Model, Tawfik and Durand (1994)) compound noise strategy would work even if extreme settings do not exist. So the conditions outlined in Theorem 4, Hou (2002) are sufficient but not necessary conditions for compound noise to work.

3.5.1 Strong Hierarchy Systems

For strong hierarchy systems the response y can be expressed as

$$y = f(x_1, x_2, ..., x_l) + \sum_{j=1}^{m} \beta_j z_j + \sum_{j=1}^{m} \sum_{i=1}^{l} \gamma_{ij} x_i z_j \qquad \textbf{(3.4)}$$

where $f(x_1, x_2, ..., x_l)$ is a general function in the control factors x_i, z_j (j=1...m) denotes noise factors affecting the system, β_j denotes effect intensity of noise factor j, and γ_{ij} denotes the effect intensities of control-by-noise interactions present in the system.

Control factors can have two settings -1 and 1. (Control factor variability can be represented as separate noise factors.) Noise factors are in the range -1 to 1, and are independent, with zero mean and variance of 1. Noise factors are symmetric about 0. The variance of y with respect to z_j's is given by

$$Var_z(y) = C + 2\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m}\beta_j\gamma_{ij}\right)x_i + \sum_{i<k}^{l}\left(\sum_{j=1}^{m}\gamma_{ij}\gamma_{kj}\right)x_i x_k\right) \tag{3.5}$$

where C is a constant. The robust setting of the system is one which minimizes the variance of response with respect to noise factors. It can be represented as

$$X \equiv \arg\left[\min_{x_i}\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m}\beta_j\gamma_{ij}\right)x_i + \sum_{i<k}^{l}\left(\sum_{j=1}^{m}\gamma_{ij}\gamma_{kj}\right)x_i x_k\right)\right] \tag{3.6}$$

where X is the setting of x_i's that minimizes the variance.

If we follow Taguchi's formulation of compound noise, based on the directionality of noise factors, then responses at two levels of compound noise will be given by

$$y_+ = f(x_1, x_2, ..., x_l) + \sum_{j=1}^{m}\left(\beta_j + \sum_{i=1}^{l}\gamma_{ij}x_i\right)sign(\beta_j) \tag{3.7}$$

$$y_- = f(x_1, x_2, ..., x_l) - \sum_{j=1}^{m}\left(\beta_j + \sum_{i=1}^{l}\gamma_{ij}x_i\right)sign(\beta_j) \tag{3.8}$$

The estimated variance with respect to compound noise levels is

$$\hat{V}ar(y) = (y_+ - \bar{y})^2 + (y_- - \bar{y})^2 = \frac{1}{2}(y_+ - y_-)^2 \tag{3.9}$$

$$\hat{V}ar(y) = C_1 + 4\left(\begin{array}{l} \sum_{i=1}^{l}\left(\sum_{j=1}^{m}|\beta_j| \sum_{j=1}^{m} sign(\beta_j)\gamma_{ij} \right) x_i + \\ \sum_{i<k}^{l}\left(\sum_{j=1}^{m} sign(\beta_j)\gamma_{ij} \sum_{j=1}^{m} sign(\beta_j)\gamma_{kj} \right) x_i x_k \end{array} \right) \tag{3.10}$$

where C_1 is a constant. The estimated robust setting of the system is one which minimizes the estimated variance of response with respect to compound noise levels. It can be represented as

$$X^{Compound} \equiv \arg\left[\min_{x_i} \left(\begin{array}{l} \sum_{i=1}^{l}\left(\sum_{j=1}^{m}|\beta_j| \sum_{j=1}^{m} sign(\beta_j)\gamma_{ij} \right) x_i + \\ \sum_{i<k}^{l}\left(\sum_{j=1}^{m} sign(\beta_j)\gamma_{ij} \sum_{j=1}^{m} sign(\beta_j)\gamma_{kj} \right) x_i x_k \end{array} \right) \right] \tag{3.11}$$

where $X^{Compound}$ is the setting of x_i's that minimizes the estimated variance. For compound noise to be completely effective the control factor setting that minimizes the estimated variance should be the same as the control factor setting that minimizes the actual variance of response, i.e.,

$$X = X^{Compound} \tag{3.12}$$

3.5.2 Weak Hierarchy Systems

For weak hierarchy systems the response can be represented by

$$y = f(x_1, x_2, ..., x_l) + \sum_{j=1}^{m} \beta_j z_j + \sum_{j=1}^{m} \sum_{i=1}^{l} \gamma_{ij} x_i z_j + \sum_{j=1}^{m} \sum_{i=1}^{l} \sum_{i<k}^{l} \theta_{ikj} x_i x_k z_j + \sum_{j=1}^{m} \sum_{j<n}^{m} \sum_{i=1}^{l} \Omega_{ijn} x_i z_j z_n \tag{3.13}$$

where y denotes system's response, $x's$ represent control factors of the system and there are l control factors present. $f(x_1, x_2, ..., x_l)$ is a general function in the control factors x_i's. β_j's denote effect intensity of noise factors. z_j's are noise factors present in the system and there are m number of noise factors. γ_{ij}'s are the effect intensities of control-by-noise interactions present in the system. θ_{ikj}'s and Ω_{ijn}'s are effect intensities of control-by-control-by-noise interactions and control-by-noise-by-noise interactions respectively, present in the system. Control factors can have two settings -1 and 1. Noise factors are in the range -1 to 1, and are independent, with zero mean and variance of 1. Noise factors are symmetric about 0. The variance of y with respect to z_j's, Rao (1992), is given by

$$Var_z(y) = C_2 + 2\left(\begin{array}{l} \sum_{i=1}^{l}\sum_{j=1}^{m}\beta_j\gamma_{ij}x_i + \sum_{i<k}^{l}\sum_{j=1}^{m}\gamma_{ij}\gamma_{kj}x_ix_k + \\ \sum_{i=1}^{l}\sum_{i<k}^{l}\sum_{j=1}^{m}\beta_j\theta_{ikj}x_ix_k + \sum_{j=1}^{m}\left(\sum_{i=1}^{l}\gamma_{ij}x_i\right)\left(\sum_{i=1}^{l}\sum_{i<k}^{l}\theta_{ikj}x_ix_k\right) + \\ \sum_{i<k}^{l}\sum_{o<i}^{l}\sum_{p<o}^{l}\sum_{j=1}^{m}\theta_{ikj}\theta_{opj}x_ix_kx_ox_p + \\ \sum_{i=1}^{l}\sum_{j=1}^{m}\sum_{j<n}^{m}\beta_j\Omega_{ijn}x_i + \sum_{i<k}^{l}\sum_{j=1}^{m}\sum_{j<n}^{m}\Omega_{ijn}\Omega_{kjn}x_ix_n + \\ \sum_{j=1}^{m}\left(\sum_{j<n}^{m}\sum_{i=1}^{l}\Omega_{ijn}x_i\right)\left(\sum_{i=1}^{l}\gamma_{ij}x_i\right) + \\ \sum_{j=1}^{m}\left(\sum_{j<n}^{m}\sum_{i=1}^{l}\Omega_{ijn}x_i\right)\left(\sum_{i=1}^{l}\sum_{i<k}^{l}\theta_{ikj}x_ix_k\right) \end{array}\right) \quad \textbf{(3.14)}$$

where C_2 is a constant. The robust setting of the system is one which minimizes the variance of response with respect to noise factors. It can be represented as

$$X \equiv \arg\left[\min_{x_i}\left(\begin{array}{l} \sum_{i=1}^{l}\sum_{j=1}^{m}\beta_j\gamma_{ij}x_i + \sum_{i<k}^{l}\sum_{j=1}^{m}\gamma_{ij}\gamma_{kj}x_ix_k + \\ \sum_{i=1}^{l}\sum_{i<k}^{l}\sum_{j=1}^{m}\beta_j\theta_{ikj}x_ix_k + \\ \sum_{j=1}^{m}\left(\sum_{i=1}^{l}\gamma_{ij}x_i\right)\left(\sum_{i=1}^{l}\sum_{i<k}^{l}\theta_{ikj}x_ix_k\right) + \\ \sum_{i<k}^{l}\sum_{o<i}^{l}\sum_{p<o}^{l}\sum_{j=1}^{m}\theta_{ikj}\theta_{opj}x_ix_kx_ox_p + \\ \sum_{i=1}^{l}\sum_{j=1}^{m}\sum_{j<n}^{m}\beta_j\Omega_{ijn}x_i + \sum_{i<k}^{l}\sum_{j=1}^{m}\sum_{j<n}^{m}\Omega_{ijn}\Omega_{kjn}x_ix_n + \\ \sum_{j=1}^{m}\left(\sum_{j<n}^{m}\sum_{i=1}^{l}\Omega_{ijn}x_i\right)\left(\sum_{i=1}^{l}\gamma_{ij}x_i\right) + \\ \sum_{j=1}^{m}\left(\sum_{j<n}^{m}\sum_{i=1}^{l}\Omega_{ijn}x_i\right)\left(\sum_{i=1}^{l}\sum_{i<k}^{l}\theta_{ikj}x_ix_k\right) \end{array}\right)\right] \quad \textbf{(3.15)}$$

where X is the settings of x_i's that minimizes the variance.

If we follow Taguchi's formulation of compound noise, based on the directionality of noise factors, then responses at two levels of compound noise will be given by

$$y_+ = f(x_1, x_2, ..., x_l) + \sum_{j=1}^{m}\left(\beta_j + \sum_{i=1}^{l}\gamma_{ij}x_i + \sum_{i=1}^{l}\sum_{i<k}^{l}\theta_{ikj}x_i x_k\right)sign(\beta_j) + \sum_{j=1}^{m}\sum_{j<n}^{m}\sum_{i=1}^{l}\Omega_{ijn}x_i sign(\beta_j)sign(\beta_n) \quad \textbf{(3.16)}$$

$$y_- = f(x_1, x_2, ..., x_l) - \sum_{j=1}^{m}\left(\beta_j + \sum_{i=1}^{l}\gamma_{ij}x_i + \sum_{i=1}^{l}\sum_{i<k}^{l}\theta_{ikj}x_i x_k\right)sign(\beta_j) - \sum_{j=1}^{m}\sum_{j<n}^{m}\sum_{i=1}^{l}\Omega_{ijn}x_i sign(\beta_j)sign(\beta_n) \quad \textbf{(3.17)}$$

The estimated variance with respect to compound noise levels is

$$\hat{V}ar(y) = C_3 + 4\left(\left(\begin{array}{l}
\sum_{i=1}^{l}\sum_{j=1}^{m}|\beta_j|\sum_{j=1}^{m}sign(\beta_j)\gamma_{ij}x_i + \sum_{i<k}^{l}\left(\sum_{j=1}^{m}sign(\beta_j)\gamma_{ij}\right)\left(\sum_{j=1}^{m}sign(\beta_j)\gamma_{kj}\right)x_ix_k + \\
\sum_{i=1}^{l}\sum_{i<k}^{l}\left(\sum_{j=1}^{m}|\beta_j|\sum_{j=1}^{m}sign(\beta_j)\theta_{ikj}\right)x_ix_k + \\
\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m}sign(\beta_j)\gamma_{ij}\right)x_i\right)\left(\sum_{i=1}^{l}\sum_{i<k}^{l}\left(\sum_{j=1}^{m}sign(\beta_j)\theta_{ikj}\right)x_ix_k\right) + \\
\sum_{i<k}^{l}\sum_{o<i}^{l}\sum_{p<o}^{l}\left(\sum_{j=1}^{m}sign(\beta_j)\theta_{ikj}\right)\left(\sum_{j=1}^{m}sign(\beta_j)\theta_{opj}\right)x_ix_kx_ox_p + \\
\sum_{i=1}^{l}\left(\sum_{j=1}^{m}|\beta_j|\left(\sum_{j=1}^{m}\sum_{j<n}^{m}sign(\beta_j)sign(\beta_n)\Omega_{ijn}\right)\right)x_i + \\
\sum_{i<k}^{l}\left(\sum_{j=1}^{m}\sum_{j<n}^{m}sign(\beta_j)sign(\beta_n)\Omega_{ijn}\right)\left(\sum_{j=1}^{m}\sum_{j<n}^{m}sign(\beta_j)sign(\beta_n)\Omega_{kjn}\right)x_ix_k + \\
\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m}\sum_{j<n}^{m}sign(\beta_j)sign(\beta_n)\Omega_{kjn}\right)x_i\right)\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m}sign(\beta_j)\gamma_{ij}\right)x_i\right) + \\
\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m}\sum_{j<n}^{m}sign(\beta_j)sign(\beta_n)\Omega_{kjn}\right)x_i\right)\left(\sum_{i=1}^{l}\sum_{i<k}^{l}\left(\sum_{j=1}^{m}sign(\beta_j)\theta_{ikj}\right)x_ix_k\right)
\end{array}\right)\right)$$

(3.18)

where C_3 is a constant. The estimated robust setting of the system is one which minimizes the estimated variance of response with respect to compound noise levels. It can be represented as

$$X^{Compound} \equiv \arg\left[\min_{x_i}\left[\begin{array}{l}
\sum_{i=1}^{l}\sum_{j=1}^{m}\left|\beta_j\right|\sum_{j=1}^{m} sign(\beta_j)\gamma_{ij}x_i + \sum_{i<k}^{l}\left(\sum_{j=1}^{m} sign(\beta_j)\gamma_{ij}\right)\left(\sum_{j=1}^{m} sign(\beta_j)\gamma_{kj}\right)x_i x_k + \\
\sum_{i=1}^{l}\sum_{i<k}^{l}\left(\sum_{j=1}^{m}\left|\beta_j\right|\sum_{j=1}^{m} sign(\beta_j)\theta_{ikj}\right)x_i x_k + \\
\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m} sign(\beta_j)\gamma_{ij}\right)x_i\right)\left(\sum_{i=1}^{l}\sum_{i<k}^{l}\left(\sum_{j=1}^{m} sign(\beta_j)\theta_{ikj}\right)x_i x_k\right) + \\
\sum_{i<k}^{l}\sum_{o<i}^{l}\sum_{p<o}^{l}\left(\sum_{j=1}^{m} sign(\beta_j)\theta_{ikj}\right)\left(\sum_{j=1}^{m} sign(\beta_j)\theta_{opj}\right)x_i x_k x_o x_p + \\
\sum_{i=1}^{l}\left(\sum_{j=1}^{m}\left|\beta_j\right|\left(\sum_{j=1}^{m}\sum_{j<n}^{m} sign(\beta_j)sign(\beta_n)\Omega_{ijn}\right)\right)x_i + \\
\sum_{i<k}^{l}\left(\sum_{j=1}^{m}\sum_{j<n}^{m} sign(\beta_j)sign(\beta_n)\Omega_{ijn}\right)\left(\sum_{j=1}^{m}\sum_{j<n}^{m} sign(\beta_j)sign(\beta_n)\Omega_{kjn}\right)x_i x_k + \\
\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m}\sum_{j<n}^{m} sign(\beta_j)sign(\beta_n)\Omega_{kjn}\right)x_i\right)\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m} sign(\beta_j)\gamma_{ij}\right)x_i\right) + \\
\left(\sum_{i=1}^{l}\left(\sum_{j=1}^{m}\sum_{j<n}^{m} sign(\beta_j)sign(\beta_n)\Omega_{kjn}\right)x_i\right)\left(\sum_{i=1}^{l}\sum_{i<k}^{l}\left(\sum_{j=1}^{m} sign(\beta_j)\theta_{ikj}\right)x_i x_k\right)
\end{array}\right]\right] \quad \textbf{(3.19)}$$

where $X^{Compound}$ is the setting of x_i's that minimizes the estimated variance. For compound noise to be completely effective $X = X^{Compound}$.

3.5.3 Conclusions from Case Studies

From the full factorial experiments done on six case studies, we found effect intensities for noise factors and control-by-noise interactions for strong hierarchy systems. For weak hierarchy systems we also found intensities of control-by-control-by-noise and control-by-noise-by-noise interactions. On plugging effect intensities for strong hierarchy systems in equations 3.6 and 3.11, we found that all three systems satisfied equation 3.12.

The effect intensities for weak hierarchy systems were plugged in equations 3.15 and 3.19 and only the Slider Crank satisfied equation 3.12. The PNM is one of the systems which do not have extreme settings, Hou (2002), but still compound noise works for such a system.

Compound noise was effective on the systems which showed *effect sparsity*, Box and Meyer (1986), Hamada and Wu (1992), Wu and Hamada (2000). For all strong hierarchy systems there were only few active control-by-noise interactions in each of the systems to be exploited during robust design experiments. The Slider Crank has only one significant noise factor to which it needs to be desensitized. The Temperature Control Circuit and CSTR did not show *effect sparsity*. These systems have *many* significant noise factors, two factor and three-factor interactions.

3.6 Conclusions

The use of compound noise in robust design experiments leads to reduction in experimental effort. Due to limitations on resources, full factorial or even fractional factorial on noise factors can't be run. Compound Noise is one of the ways in which all the noise factors can be combined in a single factor, which can be used to improve robustness of a system. In this chapter the effectiveness of compound noise as a robust design method was gaged and compared to other noise strategies. We also built upon conditions given by Hou (2002) for the complete effectiveness of compound noise. We modified the conditions to make them more general and encompass a larger set of

systems on which compound noises can work. Also those conditions were extended to include the possibility of three-factor interactions.

Compound noise as a robust design strategy is very effective on response surface instances and on systems which show *effect sparsity*. We ran compound noise on strong and weak hierarchy response surface instances with varying degree of *effect sparsity* and on six case studies from different engineering domains. The case studies were: Operational Amplifier, Passive Neuron Model, Journal Bearing, Continuous-Stirred Tank Reactor, Temperature Control Circuit and Slider Crank. Compound Noise strategy predicted robust settings for the systems which showed *effect sparsity*. It also resulted in large improvement in robustness for such systems and response surface instances (figure 3.12). The reason for its effectiveness on sparse systems is, in compound noise all the noise factors are combined. Hence their individual impact on system's response is confounded. But if effects are sparse then the probability of the impact of two noise factors being oppositely confounded is extremely low. Hence compound noise is able to exploit all significant control-by-noise interactions for such systems, leading to its high effectiveness.

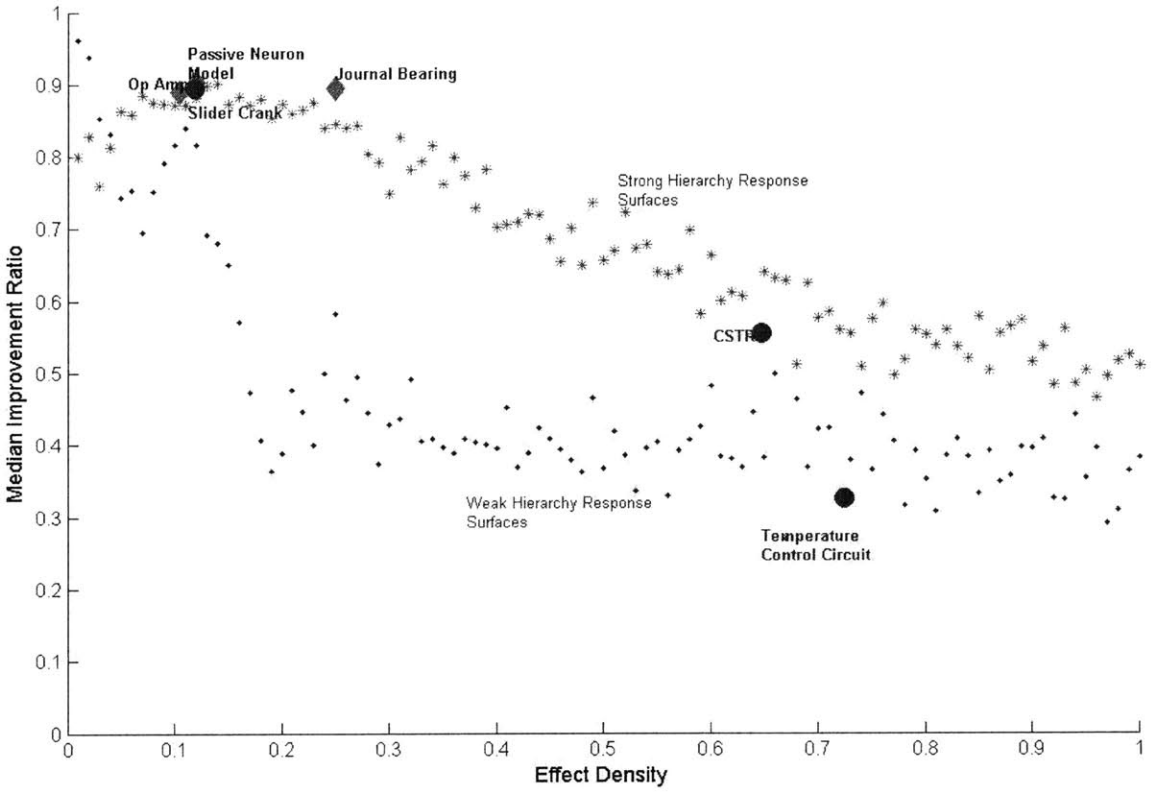


Figure 3.12: Median Improvement Ratio verses Effect Density for Strong and Weak Hierarchy Response Surface Instances and Six Case Studies

The given formulation of compound noise was adopted from Taguchi (1987) and Phadke (1989). For such a formulation the directionality of noise factors on system's response is needed to be known. To know the directionality of noise factors fractional factorial experiments on the system are needed. We tried to measure the effectiveness of simple compound noise for which the directionality of noise factors is not needed. We found that

such a formulation of compound noise can be very effective in obtaining high robustness gains. The reason why such a formulation of compound noise is very effective is that active effects in the system are sparse. Even if noise is compounded randomly there is very low probability that two opposite acting interactions get confounded with each other.

These results may be used in the overall approach to deploying compound noise as a robust design strategy. The flowchart in Figure 3.13 presents our suggestion for implementing these findings in practice. First of all, practicing engineers must define the scenario including what system is being improved, what objectives are being sought, and what design variables can be altered.

At this point, it may be possible to consider what assumptions can be made regarding *effect sparsity*. It should be noted here that we do not argue that engineers need to make a factual determination of *effect sparsity*. The experience on the system should be used to make decision.

If engineers decide that effects are dense, then they should follow the procedure on the right hand side of Figure 3.13. In this case, the results in this paper suggest that instead of using compound noise, a fractional factorial noise factor array should be used as outer array.

If engineers decide that effects are sparse, then they should follow the procedure on the left hand side of Figure 3.13. At this point, engineers should consider the directionality of noise factors. If the directionality of noise factors is known, then they should use Taguchi and Phadke's formulation of compound noise. Otherwise they can formulate simple compound noise as outer array. From Table 3.14, we can say that the systems for which *effect sparsity* holds simple compound noise is effective. However experience, judgment and knowledge of engineering and science are critical in formulating such a compound noise. It is hoped that the procedure proposed here in Figure 3.13 will be of value to practitioners seeking to implement robust design efficiently and using minimum resources.

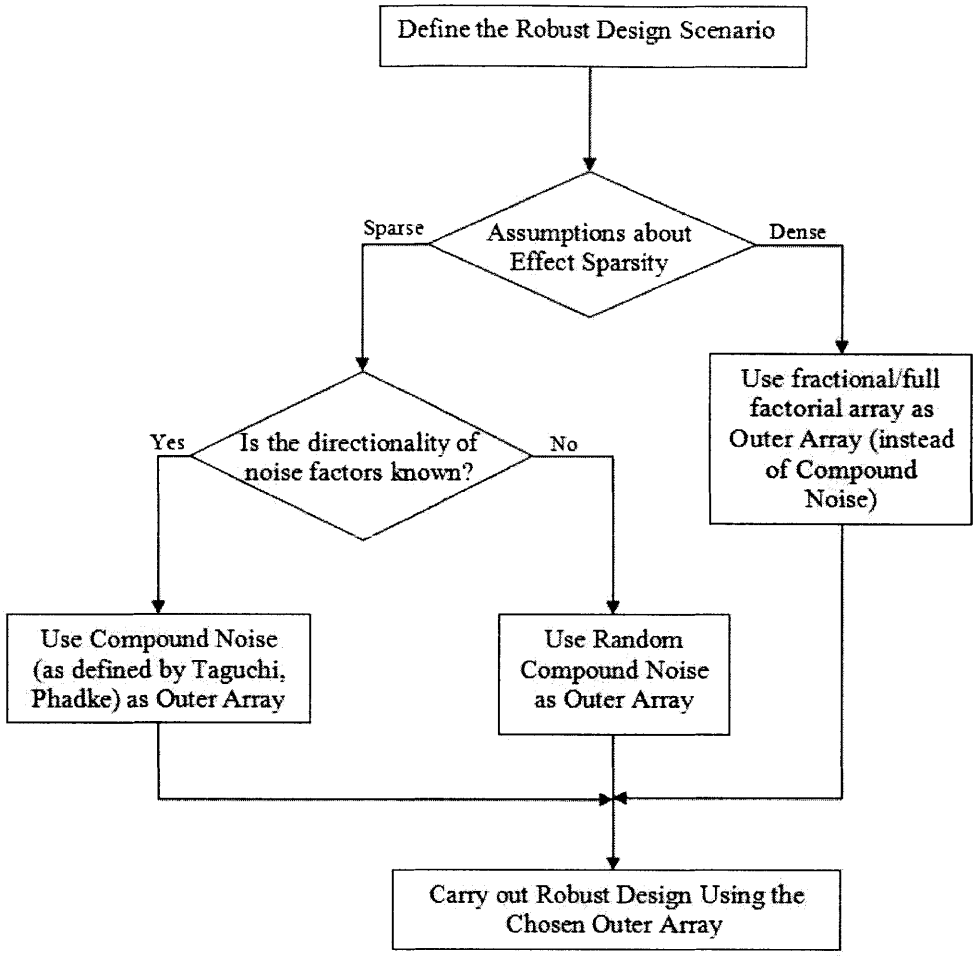


Figure 3.13: Suggested procedure for Compound Noise in Robust Design

3.7 Chapter Summary

Compound Noise as a robust design strategy, is very effective on the systems which show *effect sparsity*. The reason for its effectiveness on sparse systems is, in compound noise all the noise factors are combined. Hence their individual impact on system's response is confounded. But if effects are sparse then the probability of the impact of two noise

factors being oppositely confounded is extremely low. Hence compound noise is able to exploit all significant control-by-noise interactions for such systems, leading to its high effectiveness.

We first ran two formulations of compound noise (simple and extreme) on response surface instances generated using strong and weak hierarchical probability model. This was done to confirm compound noise effectiveness statistically. Next those formulations of compound noise were ran on six different case studies from various engineering domains to verify conclusions we got from hierarchical probability model. In the end conditions for compound noise to be completely effective for both strong and weak hierarchy systems were outlined. We engineered an algorithm on the use of compound noise as a robust design method, based on our conclusions from response surface instances and case studies.

In later chapters we will look at robust design methods which are slightly more resource intensive than compound noise, but works good for all systems. These new robust design methods are highly economical as compared to fractional factorial noise array experiments.

Chapter 4: Take-The-Best-Few Strategy: Evaluation as a Robust Design Method

4.1 Introduction and Background

To counter the effect of noise control-by-noise interactions are exploited in Robust Parameter Design methodology. These control-by-noise interactions can be captured by using crossed-array approach. The control factor setting that minimizes the sensitivity of the response to noise factors is called the optimal control factor setting or the most robust setting for the system. A crossed-array approach is a combination of two orthogonal arrays, one of control factors and other of noise factors. But as the complexity of the system increases, use of full factorial control and noise factor arrays becomes prohibitively expensive. As an attempt to reduce the run size of this crossed-array approach, Taguchi (1987) proposed a compound noise factor technique.

One of the prominent reasons why the compound noise factor strategy is so popular is that it reduces run size of experimentation needed to improve robustness. We set out to develop a noise factor strategy that retains the attractiveness of reduced run size but is more effective than the compound noise strategy. The result is the Take-The-Best-Few Noise Factors strategy (referred to hereafter as the TTBF strategy).

To apply TTBF Strategy, most important noise factors in system's noise factor space are found. Noise factors having significant impact on system response variation are

considered important. Once the important noise factors are identified for a given system, they are kept independent in the noise factor array. By selecting the few most important noise factors for a given system, run size of experiments is minimized. The TTBF strategy builds upon the TTB strategy by Gigerenzer and Goldstein (1996), which gives a criterion for selecting an alternative under uncertainty and poor information. Under the TTB strategy, a person picks one characteristic among all the available characteristics; the option that scores highest with respect to that characteristic becomes the final choice. In the TTBF strategy, instead of picking just one such characteristic, we try to choose a few such characteristics (noise factors) and determine a control factor setting that gives maximum improvement in robustness. As engineers work with a particular class of system(s), they learn about the important noise factors that affect the system(s), and this valuable knowledge is used to formulate TTBF strategy.

As in previous chapter, main aim of this chapter is to explore the effectiveness of TTBF strategy as a robust design method. We will first look at the effectiveness of TTBF strategy on response surface instances generated using Hierarchical Probability Model, Li and Frey (2005). Two different kinds of response surface instances were generated. One with only main effects and two factor interactions also called as *strong hierarchy* instances and other with main effects, two factor interactions and three factor interactions, also called as *weak hierarchy* instances. We will measure improvement in robustness while using TTBF strategy and compare it with compound noise strategy.

The conclusions from TTBF strategy on response surface instances from Hierarchical Probability Model would then be verified by testing TTBF strategy on six case studies from different engineering domains. The main take away of this chapter is that, TTBF strategy as a Robust Design Method is very effective on all response surface instances and case studies.

The chapter is organized as: first we will describe the setting up of TTBF strategy on response surface instances, measures we studied, results from TTBF strategy studies followed by discussion of these results. Then these results will be compared with the results from compound noise studies on response surface instances. Then TTBF strategy will be formulated for six case studies from different engineering domains. We will also look into the formulation of Hybrid Noise Strategy involving Compound Noise strategy and TTBF strategy. In the end conclusions and summary of the chapter will be given.

4.2 Setting up of TTBF study

The objective of this chapter is to see the effect of TTBF strategy on the prediction of optimal setting of a response surface instance generated by hierarchical probability model parameters and verify the results by case studies from various engineering domains. We also want to compare the improvement achieved by such a strategy with improvement achieved by compound noise strategies and full/fractional factorial noise array strategies. One of the assumptions in setting up of TTBF strategy is that significant noise factors impacting system's response variance are known. To find significant noise factors for a

given system or response surface instance, fractional factorial array of noise factor can be used on any setting of control factors, Phadke (1989).

4.2.1 Generating Response Surface instances

To study the effectiveness of TTBF strategy instances of response surfaces will be generated using Hierarchical Probability Model. The instances of response surfaces studied had 7 control factors and 5 noise factors. These response surfaces were generated according to a relaxed weak heredity model. The full third-order response surface equation is given by

$$y(x_1, x_2, \ldots, x_{12}) = \sum_{i=1}^{12} \beta_i x_i + \sum_{i=1}^{12} \sum_{\substack{j=1 \\ j>i}}^{12} \beta_{ij} x_i x_j + \sum_{i=1}^{12} \sum_{\substack{j=1 \\ j>i}}^{12} \sum_{\substack{k=1 \\ k>j}}^{12} \beta_{ijk} x_i x_j x_k + \varepsilon \qquad \textbf{(4.1)}$$

where x_1-x_5 represent normally distributed random noise variables, x_6- x_{12} represent control factors, $\beta' s$ are factor coefficients and ε is experimental error.

In our study, we generated both strong and weak hierarchy response surface instances. The noises factors were assumed to be uncorrelated. Full factorial array was used for 7 control factors. Table 4.1 and table 4.2 give hierarchy probability model parameter values used to generate strong and weak hierarchy response surface instances respectively.

parameters	*values*
c	15
s_1	0.33
s_2	0.67
w_2	1
p	0.43
p_{11}	0.31
p_{01}	0.04
p_{00}	0

Table 4.1: Hierarchical Probability Model Parameters used for Strong Hierarchy Response Surface instances

parameters	*values*
c	15
s_1	033
s_2	0.67
w_2	1
p	0.43
p_{11}	0.31
p_{01}	0.04
p_{00}	0.0
p_{111}	0.17
p_{011}	0.08
p_{001}	0.02
p_{000}	0.0

Table 4.2: Hierarchical Probability Model Parameters used for Weak Hierarchy Response Surface instances

For noise factors we defined twodifferent noise st rategies. First was a 2_V^{5-1} noise array. This noise strategy is very close to full factorial noise factor array. Hence it will give almost perfect results for generated response surface instances. This will form a basis of comparison for TTBF strategy. Second was TTBF strategy. In TTBF strategy for each generated response surface instance, we prioritized noise factors based on their absolute

effect coefficients $|\beta|'s$. The two noise factors (out of total five noise factors) having highest $|\beta|$ were used in TTBF strategy. Those two important noise factors were kept independent in noise array. Thus the noise array under TTBF strategy had 2^2=4 runs.

For each of the two noise strategies, we generated 200 response surface instances. For each instance of generated response surface, we used Monte Carlo on noise factors to find response variance at each level of full-factorial control factor array. The setting of control factors giving minimum variance of response was the optimal setting for that instance of response surface.

We analyzed each response surface instance by running a designed experiment using two different robust design crossed arrays: $2^7 \times 2_V^{5-1}$ (Noise Strategy 1) and $2^7 \times$ TTBF strategy. For each crossed array experiment we predicted optimal setting of control factors of response surface, by finding minimum response variance for that experiment. This predicted optimal setting of response surface under each crossed array experiment was then compared with optimal setting we got by using Monte Carlo on noise factors. In the next section we will present this algorithm in a diagrammatic manner.

4.2.2 Algorithm to study TTBF strategy

The algorithm used to study TTBF strategy is shown in Figure 4.1. The same algorithm was used to study both strong and weak hierarchy response surface instances. For each generated response surface instance, we studied two different robust design methods as

outlined in previous section. Left hand side of the algorithm finds the optimal setting of response surface instance, using Monte Carlo on noise factors. Right hand side of the algorithm finds the optimal setting of response surface instance as predicted by a chosen robust design method ($2^7 \times 2_V^{5-1}$ (Noise Strategy 1) or $2^7 \times$ TTBF strategy). Various measures are finally compared to compute the effectiveness of a chosen robust design strategy.

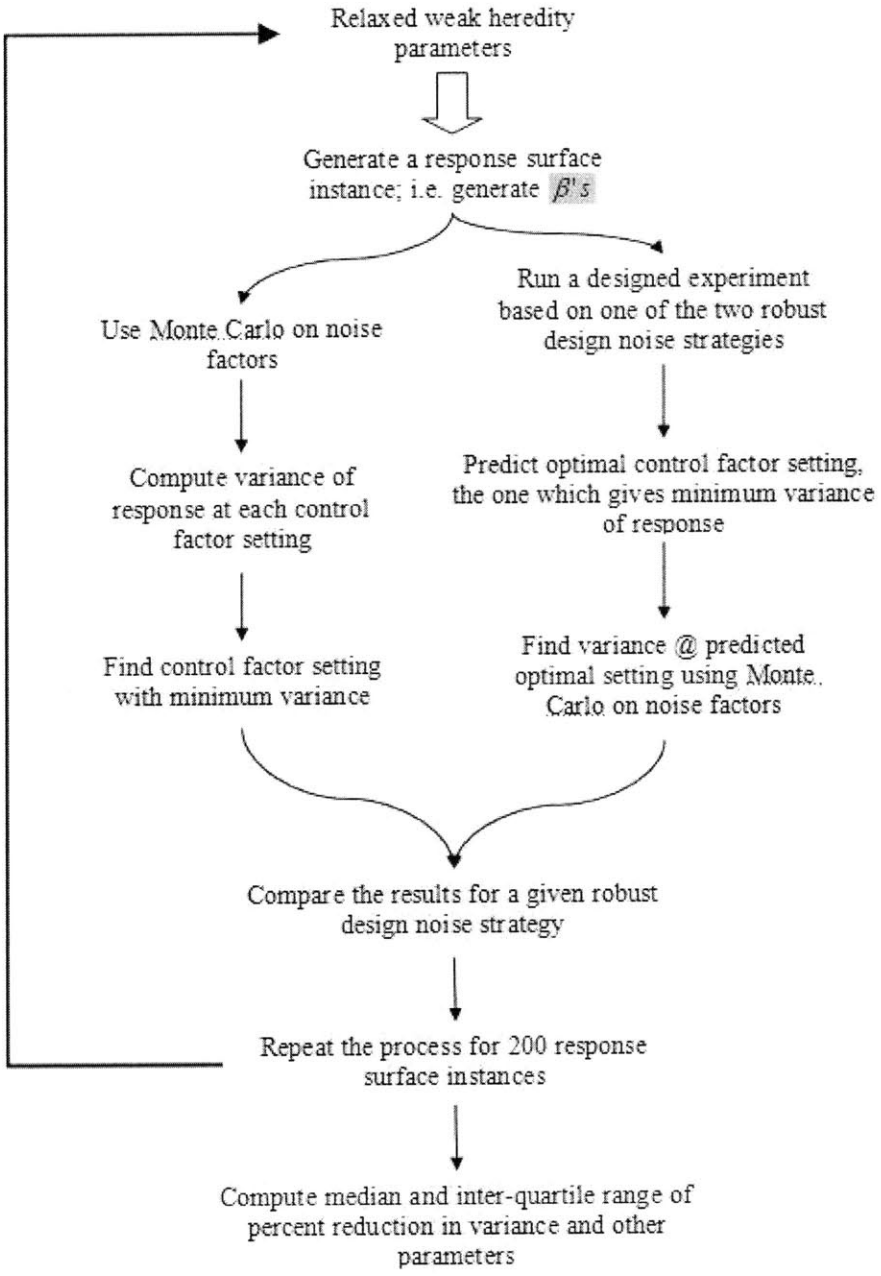


Figure 4.1: Algorithm used to evaluate TTBF strategy

4.2.3 Measures studied

There were number of measures that were derived from the above study. The first was the percentage number of times a given noise strategy could predict the optimal control factor setting, as given by running Monte Carlo simulation at each control factor setting for each generated instance of response surface.

For the second measure we found the predicted optimal settings of the 7 individual control factors as given by a noise strategy. We compared the predicted optimal setting of each control factor with the optimal control factor setting as given by Monte Carlo simulation. We then found number of control factors out the 7, which had same setting as given by noise strategy and Monte Carlo simulation. The second measure was the mean of this number for 200nstances that were generated.

For the third measure, we first found the predicted optimal control factor setting by using a given noise strategy. Then we ran Monte Carlo simulation on noise factors at that predicted optimal control factor setting and calculated the standard deviation labeled $_{strategy}$. The minimum standard deviation found by running Monte Carlo simulation at each control factor setting is called σ_{opt}. The third measure was the ratio of $\sigma_{strategy}$ to σ_{opt} for each generated instance of response surface. We studied the median and inter-quartile range of this measure for 200nstances that were generated.

For the fourth measure, we calculated the mean of the standard deviations, as given by Monte Carlo at each control factor setting. We called this mean as σ_{base}. This σ_{base} was

our reference. (σ_{base} - $_{opt}$) is the maximum reduction that is possible in standard deviation for a given instance of response surface. But by running a given noise strategy the realized reduction would only be (σ_{base} - $_{strategy}$). The fourth meaure was the ratio of realized reduction to the maximum reduction possible, i.e. ratio of (σ_{base} - $_{strategy}$) to (σ_{base} - $_{opt}$) for each generated instance. And we studied the median and inter-quartile range of this measure for 200nstances that were generated.

4.2.4 Results from TTBF Strategy studies

The results from TTBF strategy study on Strong and Weak Hierarchy response surface instances are presented from tables 4.3 to 4.8.

Percent Matching of ALL Control Factors with their Robust Setting		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
Noise Strategy 1	70%	61%
TTBF Strategy	44%	18%

Table 4.3: Measure 1

Percent Matching of Control Factors with their Robust Setting		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
Noise Strategy 1	90%	80%
TTBF Strategy	87.7%	64.3%

Table 4.4: Measure 2

Median $\left(\frac{\sigma_{strategy}}{\sigma_{opt}}\right)$		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
Noise Strategy 1	1.00	1.00
TTBF Strategy	1.00	1.15

Table 4.5: Measure 3

25th and 75th percentile $\left(\frac{\sigma_{strategy}}{\sigma_{opt}}\right)$		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
Noise Strategy 1	1.00-1.00	1.00-1.03
TTBF Strategy	1.00-1.05	1.02-1.33

Table 4.6: Measure 3

Median $\left(\frac{\sigma_{base}-\sigma_{strategy}}{\sigma_{base}-\sigma_{opt}}\right)$		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
Noise Strategy 1	1.00	1.00
TTBF Strategy	0.99	0.78

Table 4.7: Measure 4

25^{th} and 75^{th} percentile $\left(\frac{\sigma_{base} - \sigma_{strategy}}{\sigma_{base} - \sigma_{opt}}\right)$		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
Noise Strategy 1	0.98-1.00	0.95-1.00
TTBF Strategy	0.89-1.00	0.58-0.96

Table 4.8: Measure 4

To explore the impact of *effect density* on improvement in robustness that can be achieved for both strong and weak hierarchy response surface instances, we varied *effect density* over a wide range and plotted median improvement ratio for both strong and weak hierarchy response surface instances. Figures 4.2 to 4.4 present the results from such a study.

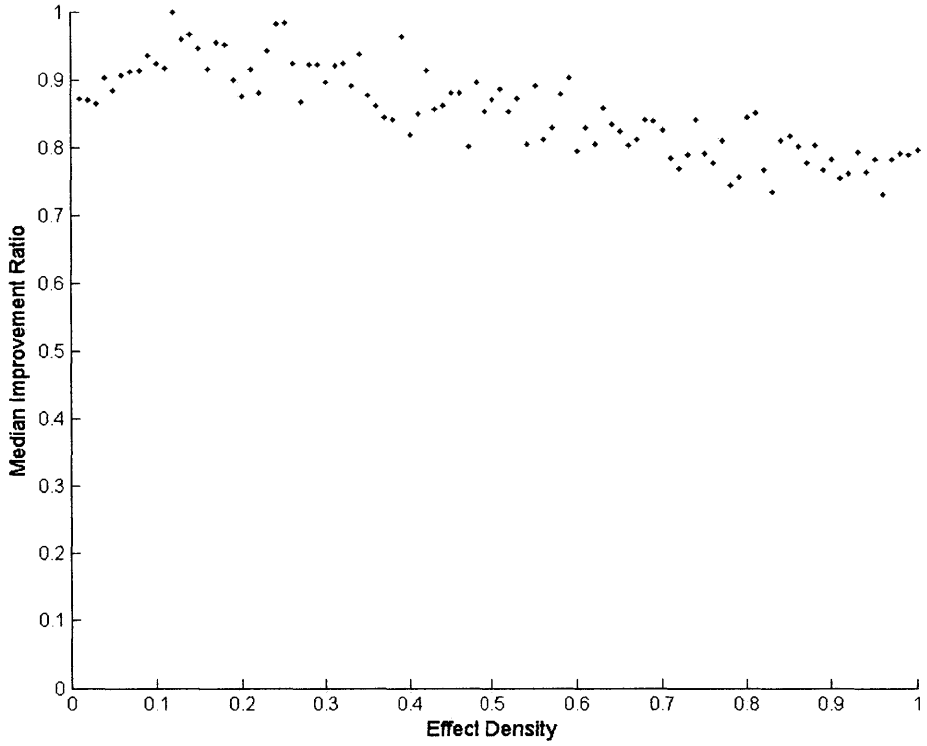


Figure 4.2: Median Improvement Ratio verses Effect Density for Strong Hierarchy Response Surface Instances

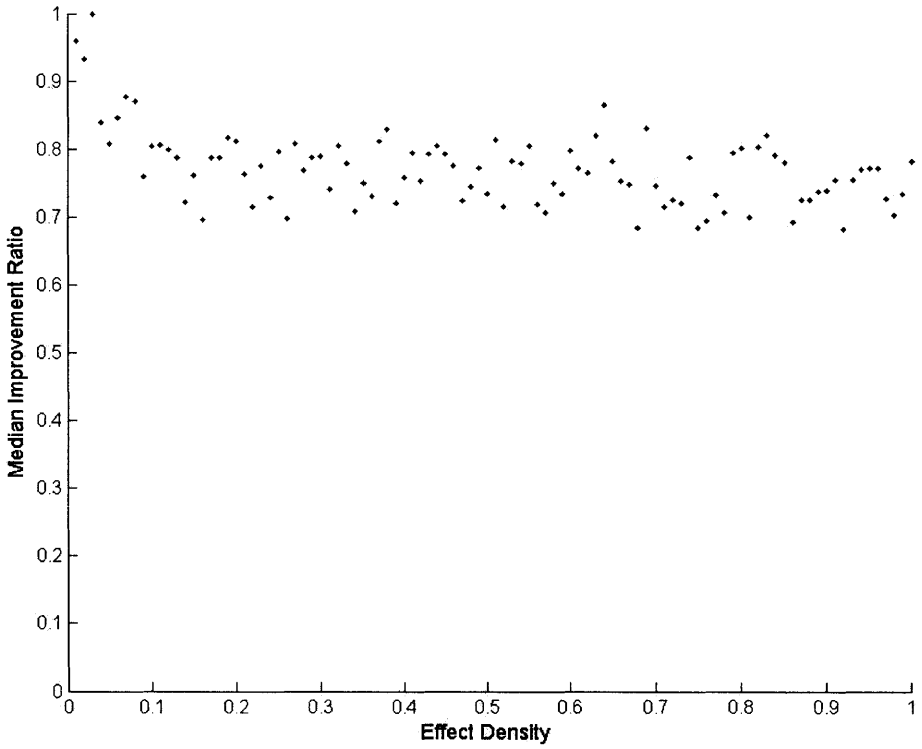


Figure 4.3: Median Improvement Ratio verses Effect Density for Weak Hierarchy Response Surface Instances

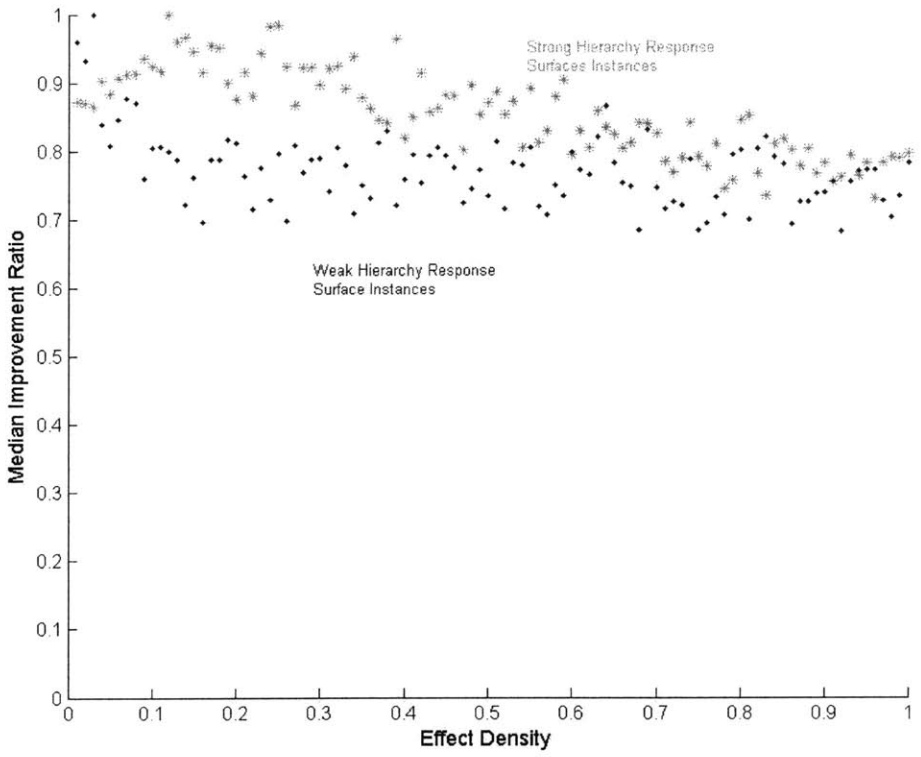


Figure 4.4: Median Improvement Ratio verses Effect Density for both Strong and Weak Hierarchy Response Surface Instances

4.2.5 Conclusions from Probability Model

The use of TTBF Strategy as robust design method leads to reduction in experimental effort. Up till now we have tried to gage the effectiveness of TTBF strategy as a robust design method for response surface instances generated using Strong and Weak Hierarchical Probability Model. As we can see from above tables and figures that TTBF strategy is very effective for both Strong and Weak hierarchical response surface instances. For both Strong and Weak hierarchical response surface instances we can

achieve on average 80% of the total possible improvement (table 4.7 and figure 4.4). In above study TTBF strategy requires only *one-eighth* of the experimental effort needed to run full factorial noise array.

Though to find two most significant noise factors out of total five noise factors, we can use a fractional factorial array for noise factors under any control factors setting. This will increase number of outer array runs on an average, but this increase is minimal as compared to overall experimental effort to run TTBF strategy for full factorial control factor array.

In the next section we will compare TTBF strategy with Compound Noise strategy. Strong and Weak hierarchical probability model response surface instances would be used as the basis for comparison of these two noise factor strategies.

4.3 Comparison of TTBF strategy and Compound Noise strategy

We generated response surface instances as per above equation 4.1 to compare TTBF and Compound Noise strategies. Each instance of generated response surface had 7 control factors and 5 noise factors. We used *extreme compound noise* as one of the noise strategies (chapter 3), Taguchi (1987) and Phadke (1989). In this strategy noise array was comprised of 2 runs, which represented lower and higher setting of compound noise. We made two-level *extreme compound noise* factor by combining noise factor levels in such a way that, at lower setting of extreme compound noise all noise factor settings give minimum response and vice versa. For the TTBF strategy, the two most important noise

factors out of the total five were found, for the generated instance of response surface and kept them independent in the noise array. Under the TTBF strategy, the noise array was comprised of 4 runs. We estimated the amount of improvement in robustness achieved using each of these two noise strategies for both strong and weak hierarchy response surface instances. Tables 4.9 to 4.14 present comparison summary of these two noise strategies for both strong and weak hierarchy response surface instances.

Percent Matching of ALL Control Factors with their Robust Setting		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
TTBF Strategy	44%	18%
Compound Noise Strategy	8%	4%

Table 4.9: Measure 1

Percent Matching of Control Factors with their Robust Setting		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
TTBF Strategy	87.7%	64.3%
Compound Noise Strategy	61%	49%

Table 4.10: Measure 2

Median $\left(\frac{\sigma_{strategy}}{\sigma_{opt}}\right)$		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
TTBF Strategy	1.00	1.15
Compound Noise Strategy	1.12	1.41

Table 4.11: Measure 3

25th and 75th percentile $\left(\frac{\sigma_{strategy}}{\sigma_{opt}}\right)$		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
TTBF Strategy	1.00-1.05	1.02-1.33
Compound Noise Strategy	1.13-1.62	1.15-1.73

Table 4.12: Measure 3

Median $\left(\frac{\sigma_{base} - \sigma_{strategy}}{\sigma_{base} - \sigma_{opt}}\right)$		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
TTBF Strategy	0.99	0.78
Compound Noise Strategy	0.69	0.43

Table 4.13: Measure 4

25th and 75th percentile $\left(\frac{\sigma_{base} - \sigma_{strategy}}{\sigma_{base} - \sigma_{opt}}\right)$		
	Strong Hierarchy response surfaces	Weak Hierarchy response surfaces
TTBF Strategy	0.89-1.00	0.58-0.96
Compound Noise Strategy	0.31-0.87	0.20-0.74

Table 4.14: Measure 4

The use of Compound Noise and TTBF strategy as robust design methods lead to reduction in experimental effort. Compound Noise strategy requires less number of experimental runs than TTBF strategy. But both require much less number of experimental runs as compared to full/fractional factorial noise strategies. We can conclude from above tables (4.9 to 4.14) that TTBF strategy is more effective than Compound Noise strategy for both Strong and Weak hierarchical response surface instances. With TTBF strategy, for both strong and weak hierarchical response surface instances on average 80% of the total possible improvement can be achieved but with Compound Noise strategy only 40% of the total possible improvement can be achieved (table 4.13).

In the next section TTBF strategy will be analyzed for six case studies from various engineering domains. We will also explore the reasons for the effectiveness of TTBF strategy on case studies.

4.4 Case Studies and effectiveness of TTBF strategy

In the previous sections we have seen that TTBF strategy works better than Compound Noise strategy in improving robustness gain for instances of response surfaces generated via strong and weak hierarchical probability model. To provide a further check on these results, we identified and implemented six system simulations, three of them followed strong hierarchy and three followed weak hierarchy. Using these six case study simulations from various engineering domains, we performed robust design studies using TTBF strategy as described in previous sections. The three case studies that exhibited strong hierarchy are Operational Amplifier (Op Amp), Phadke (1989), Passive Neuron Model, Tawfik and Durand (1994), and Journal Bearing: Half Sommerfeld Solution, Hamrock, et al. (2004). The three case studies that exhibited weak hierarchy are Continuous-Stirred Tank Reactor (CSTR), Kalagnanam and Diwekar (1997), Temperature Control Circuit, Phadke (1989) and Slider Crank, Gao, et al. (1998). For each case study we ran full factorial control and noise factor crossed array experiments. These runs were used to determine the effect coefficients for main effects, two-factor interactions and three-factor interactions. Lenth's Method, Lenth (1989), was employed to determine the active effects. The full factorial results were also used to determine the most robust setting for all six systems. Lenth Plots for case studies were given in section 3.3.1 for chapter 3.

In order to formulate TTBF strategy for robust design experiments, we found two most important noise factors, having highest impact on system's response variance for all six systems. For each system we ran a full factorial design in control factors crossed with

four run noise array, as per TTBF strategy. The two most important noise factors were kept independent in noise array and rest of the factor levels were selected randomly under TTBF strategy. This resulted in four run noise array. Pure experimental error introduced into the system's response was normally distributed with zero mean and standard deviation as the average magnitude of active main effects. TTBF strategy performed extremely well for all case studies. Table 4.15 shows result from robust design experiments.

System	Hierarchy	Measure of Sparsity of Effects	Number of Replications	Percent Matching of **All** Control Factors with their Robust Setting	Percent Matching of Control Factors with their Robust Setting	Improvement Ratio
Op Amp	Strong	0.105	100	98%	99.6%	99.96%
PNM	Strong	0.132	100	91.2%	94.6%	96.25%
Journal Bearing	Strong	0.28	100	100%	100%	100%
CSTR	Weak	0.647	100	79%	96.5%	89.74%
Temp. Control Circuit	Weak	0.725	100	71%	85.0%	80.58%
Slider Crank	Weak	0.119	100	71.4%	88.62%	85.87%

Table 4.15: Results from Full Factorial Control Factor array and TTBF strategy

The third column of Table 4.15 shows the measure of sparsity of effects, which is the ratio of active effects in a system to total number of effects that can be present in the system. The sparsity of effects means that among several experimental effects examined in any experiment, a small fraction will usually prove to be significant, Box and Meyer (1986), Hamada and Wu (1992), Wu and Hamada (2000). The fifth column of Table 4.15 shows how well the predicted robust setting from TTBF strategy matches with the actual robust setting as found by carrying out full factorial control and noise array experiments.

The sixth column of Table 4.15 shows the percent of control factors whose robust setting matched for full factorial and TTBF strategy experiments. At the predicted robust setting of control factors from TTBF strategy experiments, we found corresponding variance of system's response from full factorial experiments. Full factorial experiments yield the optimal variance of the system's response and average variance of response at all control factor settings. With this knowledge, variance can be improved from the average value to the optimal value, which is the maximum possible improvement. For TTBF strategy experiments variance of response can be improved from the average value to the variance at the predicted robust setting. The ratio of achieved improvement to maximum possible improvement is called improvement ratio. The average of this measure for all replicates is shown in the seventh column of Table 4.15.

From Table 4.15 we can see that TTBF strategy works reasonably well for all the case studies and it can achieve at least 80% of the total possible improvement in robustness for all six systems. TTBF Noise Factor strategy as a robust design strategy is very effective for all systems, even the ones which do not show *effect sparsity*. The reason for the effectiveness of TTBF strategy for all systems is that, it keeps the important noise factors in the system independent. Hence the individual impact of important noise factors on system's response is not confounded. TTBF strategy is able to exploit all significant control-by-noise interactions for such systems with very high probability, leading to its high effectiveness.

4.4.1 Effectiveness of TTBF strategy in Real scenarios

In real scenarios it is often not possible to run full factorial inner arrays like those reported on in Table 4.15. Therefore we also studied the use of fractional factorial arrays by running robust design experiments which were resolution III in both the control and noise factor array. We predicted robust settings from such experiments and compared the results to those from full factorial experiments in Table 4.16. We also ran robust design experiments which were resolution III in the control factor array combined with four run noise array (as per TTBF strategy). Results for these experiments are summarized in Table 4.17. Resolution III experiments formed the basis of comparison for the results we got from resolution III control factor array crossed with TTBF noise strategy. Pure experimental error was introduced into the system's response, for all six case studies. The error introduced was of the order of active main effects. The control factors were randomly assigned to Resolution III inner array.

We did not study Passive Neuron Model and Journal Bearing systems because these systems had only two control factors and thus had no distinct resolution III control factor array.

System	Hierarchy	Measure of Sparsity of Effects	Number of Replications	Percent Matching of **All** Control Factors with their Robust Setting	Percent Matching of Control Factors with their Robust Setting	Improvement Ratio
Op Amp	Strong	0.105	100	98%	99.6%	99.91%
CSTR	Weak	0.647	100	NONE	76.5%	96.5%
Temp. Control Circuit	Weak	0.725	100	28%	55%	87%
Slider Crank	Weak	0.119	100	91%	95.5%	97.6%

Table 4.16: Results from Resolution III Control and Noise Factor Array

Table 4.16 indicates that the robust setting for the Op Amp and Slider Crank can be achieved with high probability when a resolution III noise factor array is used. For the CSTR and Temperature Control Circuit, the high mean improvement ratio means we will not attain the robust setting but will arrive near the optima. The high improvement ratio suggests that we will achieve most of the improvement possible in the variance of system's response.

System	Hierarchy	Measure of Sparsity of Effects	Number of Replications	Percent Matching of **All** Control Factors with their Robust Setting	Percent Matching of Control Factors with their Robust Setting	Improvement Ratio	Improvement Ratio (from Resolution III Noise Array)
Op Amp	Strong	0.105	100	95%	99%	99.5%	99.91%
CSTR	Weak	0.647	100	66%	66%	87.74%	96.5%
Temp. Control Circuit	Weak	0.725	100	71%	85%	80.58%	87%
Slider Crank	Weak	0.119	100	65%	87.6%	82.71%	97.6%

Table 4.17: Results from Resolution III Control Factor array and TTBF Strategy

Table 4.17 indicates that even when using a resolution III control factor array, TTBF strategy will predict the robust setting for Op Amp, CSTR, Temperature Control Circuit and Slider Crank with high probability. TTBF strategy along with resolution III control factor array gives average improvement ratio of more than 80% for all the case studies. The reason for the effectiveness of TTBF strategy for all case studies is that, it keeps the important noise factors in the system independent. Hence the individual impact of important noise factors on system's response is not confounded. TTBF strategy is able to exploit all significant control-by-noise interactions in case studies with very high probability, leading to its high effectiveness.

4.5 Hybrid Noise Strategy

In the previous sections we analyzed the effectiveness of TTBF strategy for both response surface instances and case studies. We also have compared TTBF strategy with Compound Noise strategy for both response surface instances and case studies (Tables 3.11 and 4.15). We can infer from above sections and previous chapter that TTBF strategy performed better than Compound Noise strategy for both response surface instances and case studies. TTBF strategy needs four-run noise array whereas Compound Noise strategy needs two-run noise array. So in terms of experimental run size Compound Noise strategy is more economical than TTBF strategy.

Compound Noise as a robust design strategy is very effective on response surface instances and on systems that show *effect sparsity*. The reason for its effectiveness on sparse systems is, in compound noise all the noise factors are combined. Hence their

individual impact on system's response is confounded. But if effects are sparse then the probability of the impact of two noise factors being oppositely confounded is extremely low. On the other hand TTBF Noise Factor strategy as a robust design strategy is very effective for all systems, even the ones which do not show *effect sparsity*. The reason for the effectiveness of TTBF strategy for all systems is that, it keeps the important noise factors in the system independent. Hence the individual impact of important noise factors on system's response is not confounded. TTBF strategy is able to exploit all significant control-by-noise interactions for such systems with very high probability, leading to its high effectiveness. But TTBF strategy requires more number of experimental runs than Compound Noise strategy.

In practice robust design engineer can employ a Hybrid Noise strategy which is combination of both TTBF strategy and Compound Noise strategy. Since Compound Noise strategy works well with systems showing *effect sparsity*, this Hybrid Noise strategy will be similar to Compound Noise strategy for systems showing *effect sparsity*. Hybrid Noise strategy will be similar to TTBF strategy for rest of the systems. Thus Hybrid Noise strategy not only gives high realized reduction in the variance of system's response but also deploys noise array runs economically. For systems showing *effect sparsity* under Hybrid Noise strategy noise array will comprise of only two runs, while for rest of the systems it will comprise of four runs. Thus average number of noise array runs under Hybrid Noise strategy will be less than TTBF strategy and it would capture most of the benefit in robustness for all sort of systems.

4.6 Conclusions

Run size of experiments is one of biggest concerns while running robust design methods on any system. The reason why compound noise factor strategy is so popular is that it reduces run size of experimentation needed to improve robustness of system(s). In this chapter we set out to develop a noise factor strategy that retains the attractiveness of reduced run size but is more effective than compound noise factor strategy. We called this strategy Take-The-Best-Few Noise Factors strategy. To apply TTBF strategy most important noise factors affecting a system were found and those few noise factors were kept independent in noise factor array. In this chapter the effectiveness of TTBF strategy as a robust design method was measured and compared it with other noise strategies.

TTBF strategy as a robust design strategy is very effective for all kinds of response surface instances and systems. We ran TTBF strategy on strong and weak hierarchy response surface instances and on six case studies from different engineering domains with varying degree of *effect sparsity*. The case studies were: Operational Amplifier, Passive Neuron Model, Journal Bearing, Continuous-Stirred Tank Reactor, Temperature Control Circuit and Slider Crank. It resulted in large improvement in robustness for all systems (Table 4.15) and response surface instances (Tables 4.3-4.8 and Figure 4.5). The reason for the effectiveness of TTBF strategy for all systems is that, it keeps the important noise factors in the system independent. Hence the individual impact of important noise factors on system's response is not confounded. TTBF strategy is able to exploit all significant control-by-noise interactions for such systems with very high probability, leading to its high effectiveness.

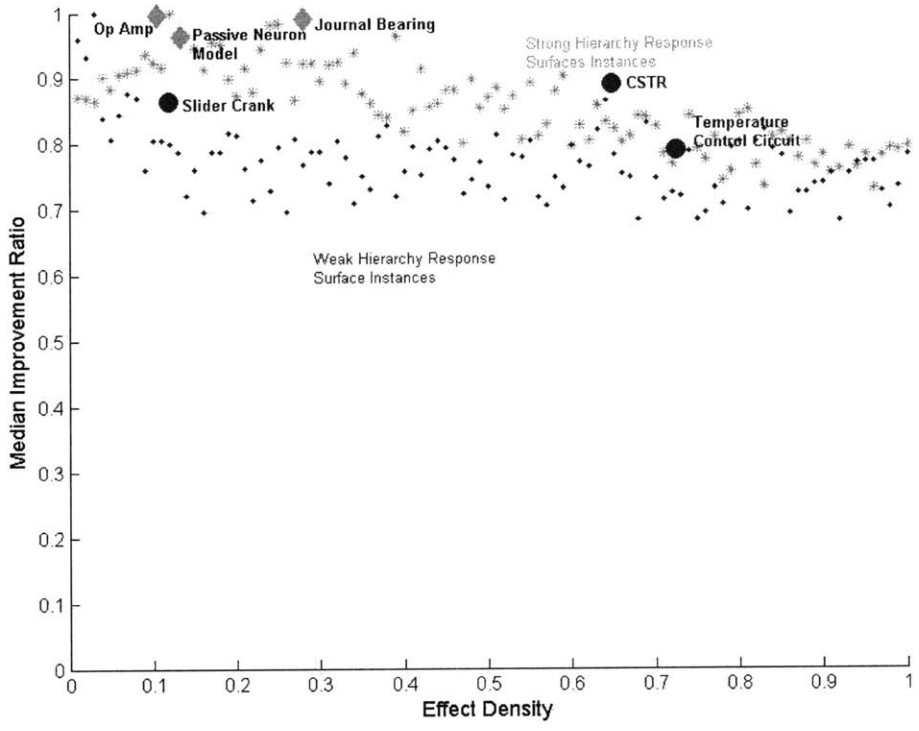


Figure 4.5: Median Improvement Ratio verses Effect Density for Strong and Weak Hierarchy Response Surface Instances and Six Case Studies

We compared TTBF strategy with Compound Noise strategy for strong and weak hierarchy response surface instances. TTBF strategy though deploys more experimental runs than Compound Noise strategy, but it is more effective. With TTBF strategy, for both strong and weak hierarchical response surface instances on average 80% of the total possible improvement can be achieved but with Compound Noise strategy only 40% of the total possible improvement can be achieved. In practice an engineer can use Hybrid Noise strategy which is similar to Compound Noise strategy for systems showing *effect*

sparsity and is similar to TTBF strategy for rest of the systems. Hybrid Noise strategy deploys noise array runs economically and gives high robustness gain for all systems.

The results of this chapter can be used in an overall approach to deploying TTBF strategy and Compound Noise strategy as a robust design strategy. The flowchart in Figure 4.6 presents our suggestion for implementing these findings in practice. This flowchart is built upon flowchart given in chapter 3 (Figure 3.13). First of all, practicing engineers must define the scenario including what system is being improved, what objectives are being sought, and what design variables can be altered.

At this point, it may be possible to consider what assumptions can be made regarding *effect sparsity*. It should be noted here that we do not argue that engineers need to make a factual determination of *effect sparsity*. The experience on the system should be used to make decision. If engineers decide that effects are sparse, then they should follow the procedure on the left hand side of Figure 4.6. But if they decide that effects are dense, then they should figure out the most important noise factors for the system under consideration. In this case they should formulate TTBF strategy by keeping all the important noise factors independent in outer array and follow the procedure on the right hand side of Figure 4.6. Figure 4.6 will be of value to practitioners seeking to implement robust design efficiently.

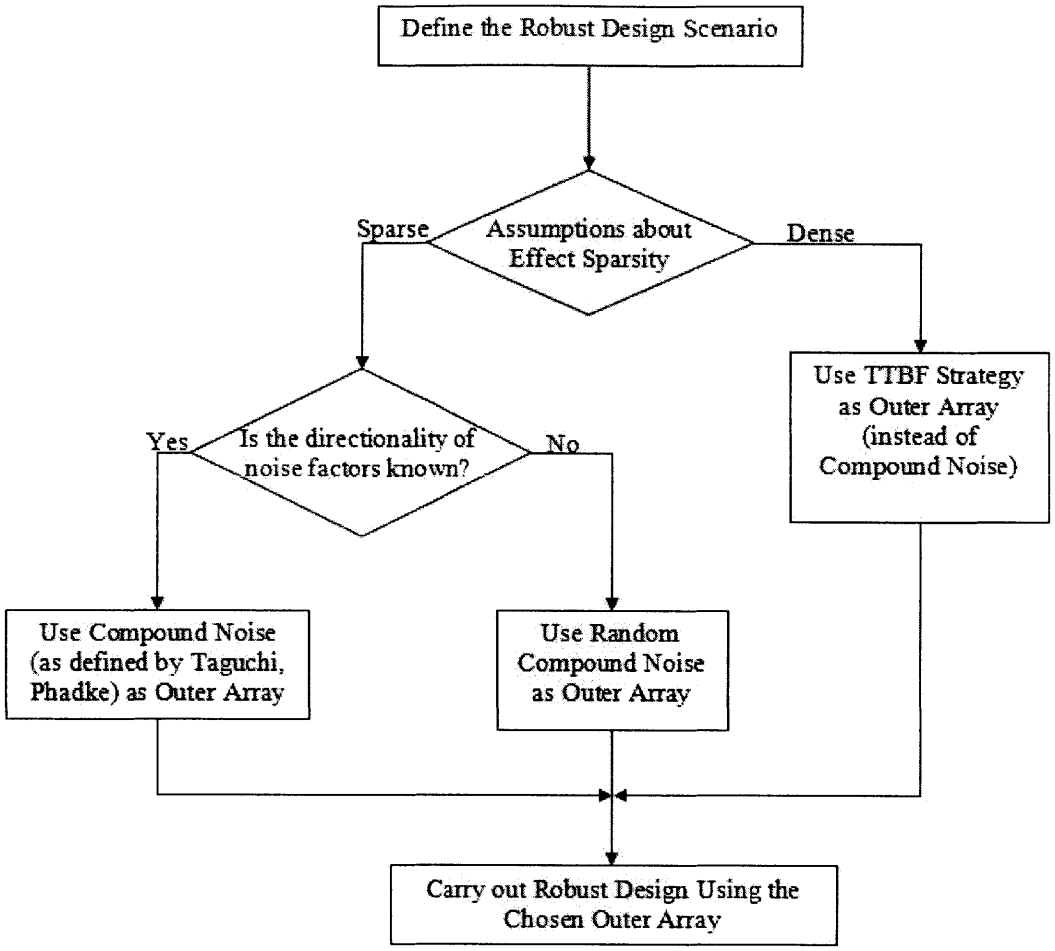


Figure 4.6: Suggested procedure for TTBF Strategy and Compound Noise in Robust Design

4.7 Chapter Summary

TTBF Noise Factor strategy as a robust design strategy is very effective for all systems, even the ones which do not show *effect sparsity*. The reason for the effectiveness of TTBF strategy for all systems is that, it keeps the important noise factors in the system

independent. Hence the individual impact of important noise factors on system's response is not confounded. TTBF strategy is able to exploit all significant control-by-noise interactions for such systems with very high probability, leading to its high effectiveness.

We ran TTBF strategy on response surfaces generated using strong and weak hierarchical probability model. This was done to confirm its effectiveness statistically. Next TTBF Noise Factor strategy was run on six different case studies from various engineering domains to verify conclusions from hierarchical probability model. We also compared TTBF strategy with Compound Noise strategy for response surfaces generated using strong and weak hierarchical probability model. We also proposed a Hybrid Noise strategy with combines the effectiveness of both TTBF strategy and Compound Noise strategy. We devised an algorithm on the use of TTBF strategy and Compound Noise strategy as robust design methods, based on our conclusions from response surface instances and case studies.

In next chapter we will look at how correlation and variance of noise factors induced during robust design influences the success of reliability improvement efforts. We will try to find a class of systems for which we can safely neglect correlation among noise factors and can amplify the magnitude of induced noise factors.

Chapter 5: Analyzing effects of correlation among and intensity of noise factors on quality of systems

5.1 Introduction and Background

Every engineering system is affected by noise factors – uncertain parameters such as environmental conditions, customer use profiles, and interface variables. Robust design methods are frequently used in industry to make subsystems and components less sensitive to noise factors. However, deployment of robust design may become too costly or time consuming unless there is an appropriate strategy focusing the effort on what is most important. This chapter documents a study to assess which aspects of noise factors have the greatest influence on the outcomes of robust parameter design. A model-based approach is presented and used to evaluate the effects of correlation among noise factors and the magnitude of noise factors on robust design. It was determined that for systems having only main effects and two-factor interactions, exaggerating the magnitude of the noise has a positive effect if pure error is high and that correlation among the noise factors has a small effect and can be safely neglected for most purposes. By contrast, for systems having active three-factor interactions, correlation among the noise factors has a large effect on the control factor settings and should be represented realistically during robust design. These results are explored through six case studies from various engineering domains.

Robust design is a set of engineering and statistical methods that can improve the consistency of a system response, Taguchi (1987). Robust design is an important part of quality/system engineering enabling smoother system integration, faster transition to production, and higher field reliability. However, robust design can also be expensive, time consuming, or otherwise resource intensive. Research that makes robust design less resource intensive is therefore of great value to engineers. As a result, many recent advances in robust parameter design have sought, for example, to reduce the number of experiments required, Wu and Hamada (2000).

In order to make a system more robust, it is generally necessary to expose the design to "noise factors" – variables such as environmental effects, manufacturing process variations, and customer usage patterns. In order to improve robustness, it is also necessary to explore alternative designs. Typically, this is done by systematically varying selected "control factors" – parameters which the designer has the latitude to change. The goal is to select a set of values of the control factors for which the system response(s) will exhibit lower variance, where variance is caused due to the noise factors.

"Noise strategy" is a term used to denote the efforts of the quality/system engineering team to characterize noise factors and to expose system to those noises. Figure 5.1 is a graphical representation of the robust design process illustrating the need for noise strategy. In this figure, a distinction is drawn between the real system that is eventually fielded and the system simulation or prototype used for design. The real system is represented by a block with two types of inputs – control factors and noise factors. The

noises that the system will actually experience in the field are denoted as "real noise". By contrast, the system simulation or prototype is exposed to "surrogate noise" – a term coined by practitioners at Ford Motor Company to describe the noises induced in system simulations or prototypes, Ejakov, Sudjianto and Pieprzak (2004). The robust parameter design process is represented by a gray box in Figure 5.1. The system simulation or prototype is exposed to the surrogate noise. Simultaneously, the control factors are varied to seek improved robustness. The process results in the selection of control factor settings.

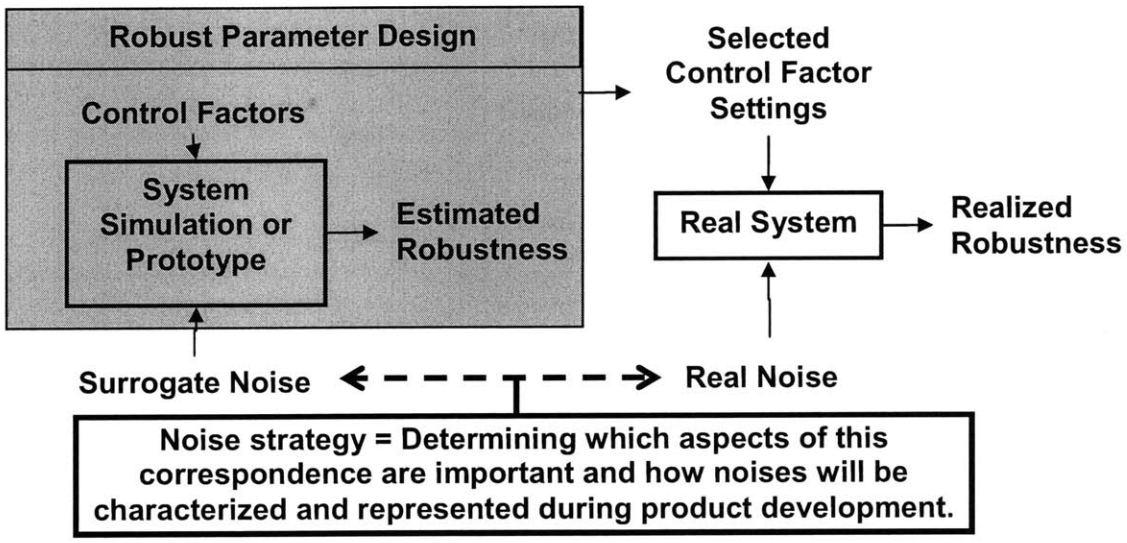


Figure 5.1: Noise Strategy in Robust Design

A key issue in noise strategy is the difference between surrogate noise and real noise. There are several reasons surrogates are not identical to real noise, of which two are important to the present investigation:

1) **The real noise is known only approximately.** Engineers are often called on to estimate the variations due to manufacture, ambient conditions, and customer use conditions. These variations can be characterized by ranges, standard deviations, correlations, and probability distributions. These characterizations are all subject to considerable ambiguity and uncertainty. Reducing that uncertainty by providing more accurate and more detailed characterization of the noises may be costly. It is particularly challenging to determine the correlation among noises in large systems. The number of correlation coefficients in a system with m noise factors is $\binom{m}{2}$. For example, in a system subjected to 15 noise factors there are over 100 correlation coefficients and in a system subjected to 46 noise factors there are over 1,000 correlation coefficients. One is tempted to neglect these correlations and focus on characterizing the marginal densities of the variables. Recent literature advises against neglecting correlation among noise factors, Goldfarb, Borror and Montgomery (2003). It is certainly true that neglecting correlation will cause some degree of error, but when robust design is applied to complex systems the costs grow to such a degree that some trade-off between cost and accuracy is likely to be necessary.

2) **The real noise conditions may be deliberately exaggerated.** Particularly when pure experimental error is high, it is often an effective strategy to induce exaggerated levels of noise so that the changes in sensitivity to noise can be more efficiently detected. This can be a highly effective strategy for increasing the information provided by a robust design experiment. For example, Joseph and Wu (2000) list noise factor amplification as one of

three feasible means of amplifying failure rates in experiments with categorical responses and they show that the amount of information from such experiments is maximized when failure rates are around 50% (which is probably much higher than what would be experienced in the field). Although the responses considered here are continuous, a similar principle applies. The efficiency of experiments might be increased by employing exaggerated noise conditions because the effects of the control factors on sensitivity are then amplified compared to measurement errors in the laboratory. On the other hand, the system may respond differently to exaggerated noises, so there is a risk inherent in this strategy.

The differences between real noise and surrogate noise may not be consequential for improving quality of a system. Ultimately, the selected control factor settings are implemented in the fielded system. The system is exposed to the real noise factors and the realized robustness of the system becomes evident. If the surrogate noise leads to the best control factor settings, the outcomes will be favorable even if the surrogate noise is different from the real noise. However, if the differences between the surrogate noise and the real noise mislead the engineering team into selecting poor control factor settings, the resulting field reliability will be less than desired.

The considerations discussed in this section led us to a set of questions to be addressed in thischapter .

- Which aspects of noise strategy have the largest influence on the realized robustness?

- Are there aspects of the real noise that one may safely neglect (at least under some conditions)?
- More specifically, are there conditions under which one may safely amplify the magnitude of noise and/or safely neglect correlation?

The main aim of this chapter is to explore the influence of correlation and variance of noise factors induced during robust design on the improvement in quality achieved for a system. We will first look at the influence on response surface instances generated using Hierarchical Probability Model, Li and Frey (2005). Two different kinds of response surface instances were generated. One with only main effects and two factor interactions also called as *strong hierarchy* instances and other with main effects, two factor interactions and three factor interactions, also called as *weak hierarchy* instances.

The conclusions from studies on response surface instances from Hierarchical Probability Model were then verified by testing similar noise strategies on six case studies from different engineering domains. The main take away from this chapter is that, systems having no significant three-factor interactions, correlation among noise factors can be safely neglected and it is often helpful to amplify the magnitude of induced noise. Otherwise, noise factors should be used that match the true magnitude and correlation in field conditions.

The chapter is organized as: first we will describe the setting up of correlation and variance study on response surface instances, measures we studied results from studies

and followed by conclusions from Hierarchical Probability Model. Then similar strategies for case studies from different engineering domains will be formulated. In the end conclusions and summary of the chapter will be given.

5.2 Setting up of correlation and variance study

The objective of this chapter is to see the influence of neglecting correlation among noise factors and exaggerating variance (intensity) of induced noise, on the prediction of optimal setting of a response surface instance. Response surface instance is generated by hierarchical probability model parameters and the results by case studies from various engineering domains will be verified. Next subsection describes a brief overview of hierarchical probability model (from chapter 2), the multiple variants of that model used in our investigation, and procedure by which the model was used.

5.2.1 Generating Response Surface instances

Hierarchical probability models have been proposed as a means to analyze the results of experiments with complex aliasing patterns. In any system we are interested in main effects and interaction effects present in the system. There is also a need to predict the relative importance and relationship among these effects. Chipman, Hamada and Wu (1997) have expressed these properties in mathematical form in a hierarchical prior probability model. The hierarchical probability model proposed by Chipman, Hamada and Wu (1997) has been extended here to enable evaluation of noise strategies in robust design. The model includes both control and noise factors since they are both needed for

the present purposes. The model includes two-factor interactions since control by noise interactions are required for robust design to be effective. It also includes the possibility of three-factor interactions since these have been shown to be frequently present, especially in systems with a large number of factors Li and Frey (2005) and might affect the outcomes of robust design. The details of the model are in Equations 5.1 through 5.10.

$$y(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \beta_{ij} x_i x_j + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j>i}}^{n} \sum_{\substack{k=1 \\ k>j}}^{n} \beta_{ijk} x_i x_j x_k + \varepsilon \tag{5.1}$$

$$x_i \sim N(0, \mathbf{K}) \; i \in 1 \ldots m \tag{5.2}$$

$$x_i \in \{+1, -1\} \; i \in m+1 \ldots n \tag{5.3}$$

$$\varepsilon \sim NID(0, \sigma_\varepsilon^{\,2}) \tag{5.4}$$

$$f(\beta_i | \delta_i) = \begin{cases} N(0,1) & \text{if } \delta_i = 0 \\ N(0, c^2) & \text{if } \delta_i = 1 \end{cases} \tag{5.5}$$

$$f(\beta_{ij} | \delta_{ij}) = \begin{cases} N(0,1) & \text{if } \delta_{ij} = 0 \\ N(0, c^2) & \text{if } \delta_{ij} = 1 \end{cases} \tag{5.6}$$

$$f(\beta_{ijk}|\delta_{ijk}) = \begin{cases} N(0,1) & \text{if } \delta_{ijk} = 0 \\ N(0,c^2) & \text{if } \delta_{ijk} = 1 \end{cases} \tag{5.7}$$

$$\Pr(\delta_i = 1) = p \tag{5.8}$$

$$\Pr(\delta_{ij} = 1|\delta_i, \delta_j) = \begin{cases} p_{00} & \text{if } \delta_i + \delta_j = 0 \\ p_{01} & \text{if } \delta_i + \delta_j = 1 \\ p_{11} & \text{if } \delta_i + \delta_j = 2 \end{cases} \tag{5.9}$$

$$\Pr(\delta_{ijk} = 1|\delta_i, \delta_j, \delta_k) = \begin{cases} p_{000} & \text{if } \delta_i + \delta_j + \delta_k = 0 \\ p_{001} & \text{if } \delta_i + \delta_j + \delta_k = 1 \\ p_{011} & \text{if } \delta_i + \delta_j + \delta_k = 2 \\ p_{111} & \text{if } \delta_i + \delta_j + \delta_k = 3 \end{cases} \tag{5.10}$$

This hierarchical probability model allows any desired number of response surface instances to be created such that population of response surface instances has the desired properties of sparsity of effects, hierarchy, and inheritance. Equation 5.1 represents the measured response of the engineering system *y*. The independent variables x_i's are both control factors and noise factors. Control and noise factors are not distinguished in this notation except via indices. Equation 5.2 shows that the first set of *m* input parameters to Hierarchical Probability Model ($x_1, x_2, \ldots, x_m$) are regarded as noise factors and are assumed to be normally distributed with variance-covariance **K** among noise factors. Equation 5.3 shows that the other input variables ($x_{m+1}, x_{m+2}, \ldots x_n$) are the control factors which are assumed to be two-level factors. The variable represents the pure

experimental error in the observation of response which is assumed to be normally distributed.

The model response y is assumed to be a third order polynomial of the input variables x_i. The values of the polynomial coefficients (β's) are determined by a process implemented with pseudo-random number generators. Equation 55 determines the probability density function for the first order coefficients. Factors can be either "active" or "inactive" depending on the value (0 or 1 respectively) of their corresponding parameters . The strength of active effects is assumed to be c times that of inactive effects. Equations 56 and 57 determine the probability density function for second order and third order coefficients respectively.

Equation 58 sets the probability p of any main effect being active. Equation 59 and 510 enforce inheritance. If the parameters are set so that $p_{11}>p_{01}>p_{00}$ and so on, then the likelihood of any second order effect being active is lowest if no participating factor has an active main effect and is highest if all participating factors have active main effects.

Given the model in Equations 51 through 510, some inferences about noise strategy can be made based on probability theory. The following formula adapted from Siddall (1983) gives the transmitted variance in y (as given in Equation 51) assuming all terms of the form $_{ijk}$=0

$$\sigma^2(y)=\sum_{i=1}^{m}\left(\beta_i+\sum_{j=m+1}^{n}\beta_{ij}x_j\right)^2\mathrm{K}_{ii}+2\sum_{i=1}^{m}\sum_{j=1}^{i-1}\left(\beta_i+\sum_{k=m+1}^{n}\beta_{ik}x_k\right)\left(\beta_j+\sum_{k=m+1}^{n}\beta_{jk}x_k\right)\mathrm{K}_{ij}+$$
$$2\sum_{i=1}^{m}\sum_{\substack{j=1\\j>i}}^{m}\sum_{\substack{k=1\\k>j}}^{m}\left(\beta_i+\sum_{p=m+1}^{n}\beta_{ip}x_p\right)\beta_{jk}\left(\mathrm{K}_{ij}+\mathrm{K}_{ik}\right)+2\sum_{i=1}^{m}\sum_{\substack{j=1\\j>i}}^{m}\sum_{\substack{k=1\\k>j}}^{m}\left(\beta_j+\sum_{p=m+1}^{n}\beta_{jp}x_p\right)\beta_{ik}\left(\mathrm{K}_{ij}+\mathrm{K}_{jk}\right)+$$
$$+2\sum_{i=1}^{m}\sum_{\substack{j=1\\j>i}}^{m}\sum_{\substack{k=1\\k>j}}^{m}\left(\beta_k+\sum_{p=m+1}^{n}\beta_{kp}x_p\right)\beta_{jk}\left(\mathrm{K}_{ik}+\mathrm{K}_{jk}\right)$$

(5.11)

This equation shows that off-diagonal terms in the covariance matrix (terms of the form $\mathbf{K}_{ij}$) do influence transmitted variance in a response due to noise. However, these terms will not be a function of the control factors if all the noise by noise interactions are zero (if terms of the form $_{jk}$=0 where j and k are less than or equal to m). Therefore, probability theory suggests that if three-factor interactions can be neglected and noise by noise interactions are small, then robust design will not be strongly affected by correlation among noise factors. No similar statement can be made if three-factor interactions might be large. Closed form equations can only provide these rather qualitative statements; therefore further results will be pursued via simulations.

Multiple variants of the hierarchical probability model were formed by selecting different sets of model parameters as described in Table 5.1. As the column headings of Table 5.1 indicate, a key difference between the variants is the assumption concerning effect hierarchy. A Strong Hierarchy Model assumes that the only active effects in the system are main effects and two-factor interactions although small three factor interactions are

present as can be seen in Equation 5.7. A Weak Hierarchy Model includes a possibility for active three-factor interactions. The values for parameters in Table 5.1 such as p_{1I}=0.25 and p_{0I}=0.1 are based on the Weak Heredity model proposed by Chipman, Hamada and Wu (1997). In fact, the Strong Hierarchy model is precisely the Weak Heredity model published in that paper and used for analyzing data from experiments with complex aliasing patterns. The Weak Hierarchy model proposed here is an extension of that model to include higher order effects and therefore relies less on the assumption of hierarchy.

Additional model variants are based on the options in the last three rows of Table 5.1. The on-diagonal elements of the covariance matrix were varied among two levels. The covariance matrix was also composed by three different methods inducing different degrees of correlation. These resulted in off-diagonal elements of the covariance matrix with different average magnitudes. Given the two model options related to the columns of Table 5.1 and the additional combinations of options due to the alternatives in the last three rows, there are 24 different model variants in all.

parameters	**Strong Hierarchy Model (active main effects and two-factor interactions)**	**Weak Hierarchy Model (active three-factor interactions also included)**
m	5	5
n	12	12
c	10	10
p	0.25	0.25
p_{11}	0.25	0.25
p_{01}	0.1	0.1
p_{00}	0.0	0.0
p_{111}	0.0	0.25
p_{011}	0.0	0.1
p_{001}	0.0	0.0
p_{000}	0.0	0.0
σ_ε	1 or 10	1 or 10
$\mathbf{K}_{ii}$	1.0 or 1.75	1.0 or 1.75
$\mu(\mathbf{K}_{ij} \mid)$, $i \neq j$	0.01, 0.26, or 0.47	0.01, 0.26, or 0.47

Table 5.1: Parameters for Variants of Hierarchical Probability Model

K is the variance-covariance matrix for the real noise factors. The modeled noise, in response surface instance is assumed to have a covariance of an identity matrix. Thus the more different **K** is from an identity matrix, the more the noise strategy varies from a faithful representation of the noises the system will experience in the field.

The on-diagonal elements of the matrix, $\mathbf{K}_{ii}$, are the variance due to each noise factor x_i. The size of these on-diagonal elements is an indication of the amplitude of the real noise factors relative to the modeled noise factors. Two options within the model are defined: one in which the real noise has the same variance as the modeled, and one in which the real noise has higher variance than the modeled.

The off-diagonal elements of the matrix, $\mathbf{K}_{ij}$, are the covariance among noise factors x_i and x_j. Three options within the model are defined: one with almost no correlation (with the average absolute value of the correlation coefficients being 0.01), one with relatively mild correlation (with the average absolute value of the correlation coefficients being 0.26) and one with relatively strong correlation (with the average absolute value of the correlation coefficients being 0.47). The matrix **K** was formed so as to ensure the resulting matrix was positive semi-definite while also having the desired variance and the desired degree of correlation.

5.2.2 Algorithm to evaluate noise strategies

The different variants of the hierarchical probability model were used to evaluate noise strategies as described in Figure 52. The first step in the algorithm is to generate 1000 instances of the Strong Hierarchy Model and 200 instances of the Weak Hierarchy Model. Fewer instances of the Weak Hierarchy Model are used because that model takes much longer to run given that it contains 220 (12 choose 3) extra polynomial terms to evaluate for each observation of the response. For all of these model instances, a full factorial experiment is crossed with Latin Hypercube sampling assuming the variance-covariance matrix is an identity matrix. The control factor settings that minimize the transmitted variance are recorded. This simulates the choice of control factor settings under a noise factor surrogate. The same process is repeated for six different versions of the variance-covariance matrix **K**. For each generated instance of this matrix **K**, the average transmitted variance, the minimum transmitted variance, and the control factor

settings that minimize the transmitted variance are recorded. In addition, the previously recorded control factor settings that minimize the transmitted variance assuming **I,** are used to look up the transmitted variance assuming **K**. This step simulates a confirmation experiment under the real noise conditions given the settings selected under the surrogate noise conditions.

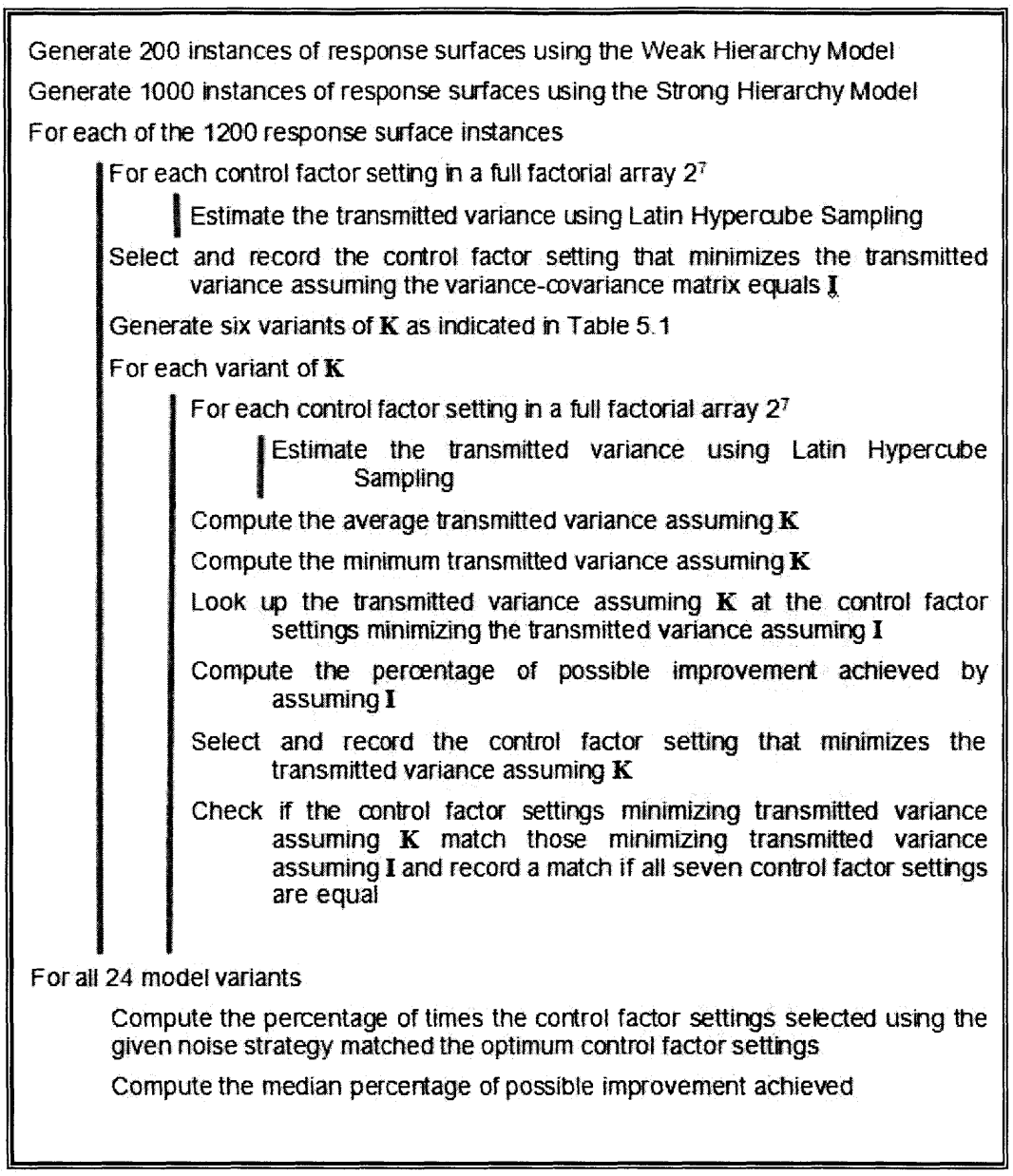


Figure 5.2: Algorithm used to evaluate Noise Strategies

5.2.3 Measures studied

Through the algorithm in Figure 52, two different performance measures are computed:

1) Median fraction of possible reduction in standard deviation. At any particular setting of the control factors, a response surface will exhibit a given amount of variance due to variation of the input noise factors. The most robust of all the control factor settings will exhibit the least variance and have the lowest standard deviation of the response. The possible reduction in standard deviation is the difference between the minimum standard deviation and the average standard deviation throughout the possible settings of the control factors. A robust design process will rate 1.0 if it always attains the lowest standard deviation possible. Selection of control factor settings at random would rate 0.0. Most robust design strategies will earn a rating in between these two extremes.

2) Percent of systems for which optimum control factor settings matched. A goal of robust design is to select the best settings of the control factors. However, any robust design method is likely to occasionally miss the optimal setting of at least one factor. The average number of cases in which the exact optimum settings are attained is another performance measure for a noise strategy.

5.2.4 Results from model-based analysis

The procedure described in the previous section was carried out. Table 5.2 presents the results with the performance metric defined as the median fraction of maximum

improvement attained. Table 53 presents the results with the performance metric defined as the probability of attaining the optimum control factors.

				Strong Hierarchy Model (active main effects and two-factor interactions)					Weak Hierarchy Model (active main effects, two-factor, and three-factor interactions)				
					Correlation					Correlation			
				None	Mild	Strong			None	Mild	Strong		
Pure Experimental Error	Large error $\varepsilon \sim NID(0,10^2)$	Noise	Matching	0.97	0.89	0.81	Noise	Matching	0.94	0.84	0.70		
			Amplified	1.00	0.92	0.86		Amplified	0.90	0.81	0.56		
	Small error $\varepsilon \sim NID(0,1^2)$	Noise	Matching	1.00	0.96	0.90	Noise	Matching	1.00	0.89	0.72		
			Amplified	1.00	0.96	0.90		Amplified	0.92	0.82	0.57		

Table 5.2: Median fraction of the maximum possible improvement attained in hierarchical probability response surface instances

				Strong Hierarchy Model (active main effects and two-factor interactions)					Weak Hierarchy Model (active main effects, two-factor, and three-factor interactions)				
					Correlation					Correlation			
				None	Mild	Strong			None	Mild	Strong		
Pure Experimental Error	Large error $\varepsilon \sim NID(0,10^2)$	Noise	Matching	36%	20%	13%	Noise	Matching	38%	21%	9%		
			Amplified	49%	26%	13%		Amplified	32%	16%	11%		
	Small error $\varepsilon \sim NID(0,1^2)$	Noise	Matching	81%	32%	13%	Noise	Matching	57%	36%	21%		
			Amplified	72%	18%	11%		Amplified	40%	35%	18%		

Table 5.3: Percentage of hierarchical probability response surface instances in which optimum control factor settings were attained

5.2.5 Significance of results from model-based approach

Both Tables 52 and 53 suggest that increased correlation in the noise f actors adversely affects performance of robust design if that correlation is not represented in the noise strategy. It is also salient that correlation among the noise factors has a much stronger effect on robust design if the Weak Hierarchy Probability model is assumed than if the Strong Hierarchy Probability model is assumed. It is also noteworthy that increased correlation generally had a milder effect on the median fraction of the optimum reduction in standard deviation than it did on percentage matching control factor settings. There is only one optimum setting of the control factors, but there can be many settings that provide results nearly as good as the optimum. It appears that lack of fidelity in modeling correlation leads to selection of settings that are only slightly sub-optimal. Conclusions about the need to model correlation in noise strategy depend critically on the goals of the project. One is more likely to be able to neglect correlation if improved consistency of the response is the goal and failing to attain the optimum settings of control factors can be tolerated.

The magnitude of the noise factors has a mild effect on the performance of robust design, except when the system has active three factor interactions and the correlation is moderate or high. Note also that the effect of amplifying the noise is positive if the pure experimental error is high and strong hierarchy can be assumed. However, if only weak hierarchy can be assumed, then the response to the noise depends on the amplitude of the noise and it is probably better to induce noises typical of field levels rather than exaggerated. However, if those levels are not known, exaggerating the noise amplitude

will have only mild negative effects on the design process. To summarize, under most conditions, it appears to be safe to amplify the noise and under selected conditions amplifying noise can have a beneficial effect.

In next section we will analyze correlation and variance effect of induced noise on six case studies from various engineering domains. We will compare results from case studies with results we got from hierarchical probability response surface instances.

5.3 Case Studies

In the previous section we inferred that the choice of noise strategy depends on assumptions about hierarchy – that is, whether three-factor interactions can be assumed to be small. To provide a further check on these results, we identified and implemented six system simulations; three that contain significant three-factor interactions and three that do not. Three case studies that do not contain significant three-factor interactions are Operational Amplifier (Op Amp), Phadke (1989), Passive Neuron Model (PNM), Tawfik and Durand (1994), and Journal Bearing: Half Sommerfeld Solution, Hamrock, et al. (2004). Three case studies that contain significant three-factor interactions are Continuous-Stirred Tank Reactor (CSTR), Kalagnanam and Diwekar (1997), Temperature Control Circuit, Phadke (1989) and Slider Crank, Gao, et al. (1998). Using these six case studies, we performed simulated robust design studies using various noise strategies as described in the last section. By this means, we were able to create results similar in structure to those in Tables 5.2 and 5.3 but for case studies rather than for a hierarchical probability model response surface instance.

In case studies pure experimental error was comparable to that of one-tenth of largest noise factor intensity. We can gage noise factor intensities for case studies from Lenth Plots (Figures 3.6 to 3.11). Each noise strategy was replicated in order to assess the repeatability of the results.

5.3.1 Results from Case Studies

Tables 5.4 and 5.5 describe the results we obtained from case studies.

System	Hierarchy			Correlation		
				None	Mild	High
Op Amp	Strong	Noise Intensity	Matching	1.00	1.00	1.00
			Amplified	1.00	1.00	1.00
PNM	Strong	Noise Intensity	Matching	1.00	1.00	1.00
			Amplified	1.00	1.00	1.00
Journal Bearing	Strong	Noise Intensity	Matching	1.00	1.00	1.00
			Amplified	1.00	1.00	1.00
CSTR	Weak	Noise Intensity	Matching	0.99	0.96	0.70
			Amplified	0.71	0.51	0.35
Temperature Control Circuit	Weak	Noise Intensity	Matching	1.00	0.96	0.92
			Amplified	0.95	0.85	0.73
Slider Crank	Weak	Noise Intensity	Matching	1.00	0.99	0.99
			Amplified	1.00	0.98	0.96

Table 5.4: [1]Median fraction of the maximum improvement attained in case study simulations

[1] With replications

System	Hierarchy				Correlation	
				None	Mild	High
Op Amp	Strong	Noise Intensity	Matching	100%	100%	100%
			Amplified	100%	100%	100%
PNM	Strong	Noise Intensity	Matching	100%	100%	100%
			Amplified	100%	100%	100%
Journal Bearing	Strong	Noise Intensity	Matching	100%	100%	100%
			Amplified	100%	100%	100%
CSTR	Weak	Noise Intensity	Matching	75%	50%	25%
			Amplified	25%	12.5%	0%
Temperature Control Circuit	Weak	Noise Intensity	Matching	100%	89%	84%
			Amplified	87%	74%	66%
Slider Crank	Weak	Noise Intensity	Matching	100%	95%	94.7%
			Amplified	100%	92.5%	86%

Table 5.5: Percentage of case study simulations in which optimum control factor settings were attained

The second column of tables 5.4 and 5.5 shows whether case study follow strong hierarchy model (i.e. no significant three-factor interactions are present)v or follow weak hierarchy model. The third and fourth column show whether induced noise intensities matched with actual noise levels or were amplified as compared to actual noise levels. The last three columns capture the impact of correlation among actual noise and the error we will make in neglecting that correlation for induced noises. We can infer from these tables that correlation among the noise factors has a much stronger effect on robust design if case studies follow weak hierarchy model than if they follow strong hierarchy model. Also increased correlation generally has milder effect on the median fraction of

the optimum reduction in standard deviation than it did on percentage matching of optimal control factor settings. There is only one optimum setting of the control factors, but there can be many settings that provide results nearly as good as the optimum.

5.4 Conclusions

Carrying out robust design requires, either implicitly or explicitly, a noise strategy. One must decide how to represent the noise factors the product is likely to encounter during its lifecycle. Only then can the product be exposed to those noise factors while searching for design changes that improve robustness. Designers often have only a rough estimate of the conditions the product will actually encounter and therefore must make assumptions. It is useful therefore to understand what assumptions can be made safely and which assumptions, if violated, will lead to disappointing results.

Using a hierarchical probability model response surface instances and following up with case studies, we found that:

- With a strong assumption of hierarchy, surrogate noises can safely be used in robust design that neglect correlation and/or exaggerate the intensity of noise factors.
- With a strong assumption of hierarchy and in the presence of large experimental errors, surrogate noises that exaggerate the intensity of noise factors may improve the results of robust design.
- Absent a strong assumption of hierarchy, correlation should **not** be neglected in the noise strategy and noise factors should **not** be amplified.

These results may influence one's strategy in deploying robust design in product development. The flow chart in Figure 53 presents our suggestion for implementing these findings in practice. First of all, the engineering team must define the scenario including what system is being improved, what objectives are being sought, and what design variables may be altered. After defining the parameters of the scenario, it should be possible to assess the assumptions can be made regarding effect hierarchy. We are not suggesting that the team needs to make a factual determination of the existence of three-factor interactions. The experience of the team should be used to make an assessment which will necessarily be made under uncertainty.

If the team decides that only weak hierarchy can be assumed, that is, that three-factor interactions might be present, then the team should choose the procedure on the right hand side of Figure 5.3. In this case, the team must decide whether it is possible or advisable to take measures that eliminate or reduce the possibility of three-factor interactions. In some cases, reformulation of the response, or the input variables, or redesign of the system can be effective. It is possible that formulating responses related to the energy transformations in the system will greatly reduce the likelihood of high order interactions. If the team finds it difficult to avoid the possibility of such interactions, the results in this chapter suggest that the noise strategy should seek to accurately represent both the intensity and correlations of the noises.

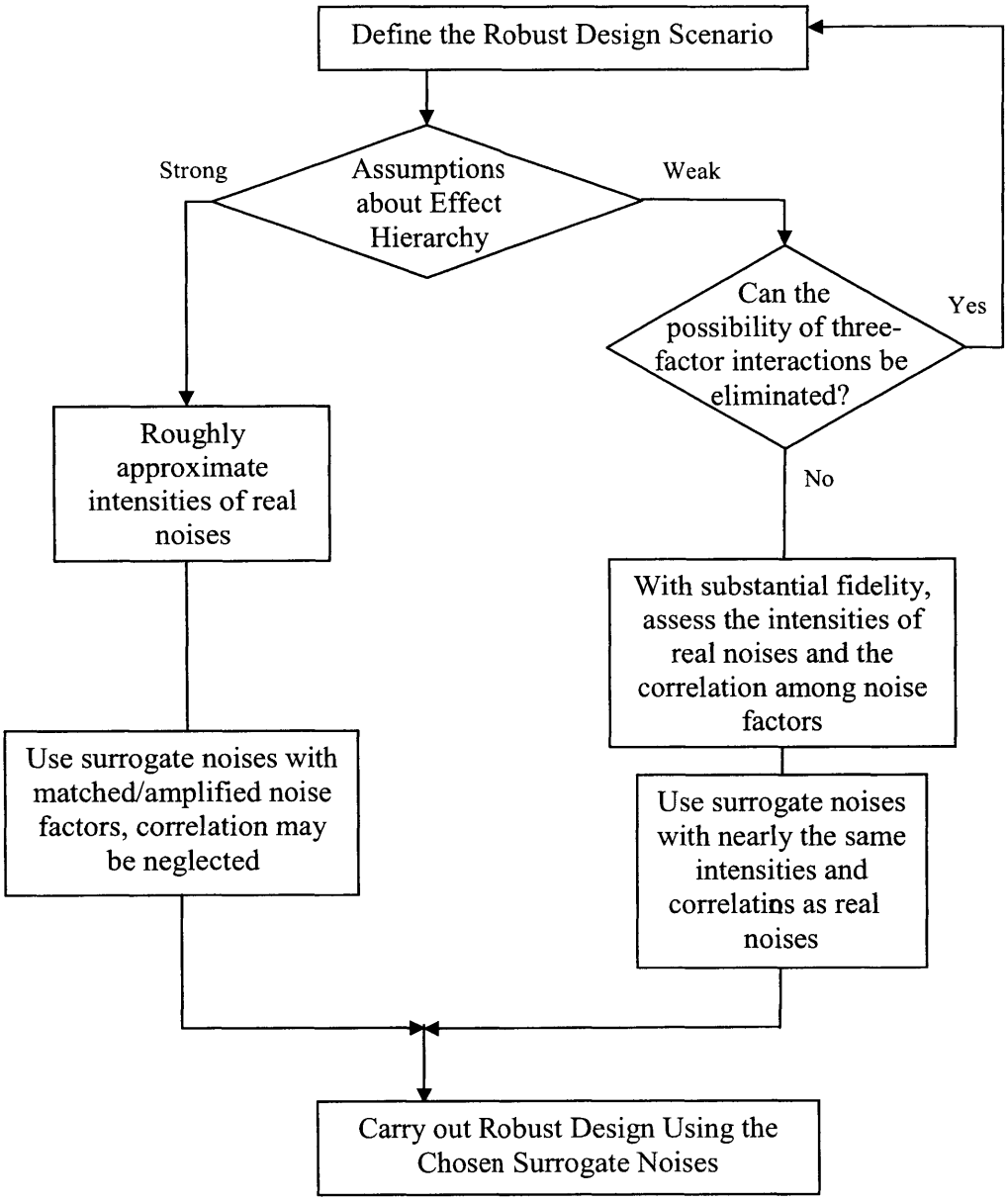


Figure 5.3: Flowchart to reduce information needed to implement Robust Design Methodology

If the type of system under consideration, the type of response, and the design variables are judged as unlikely to exhibit three-factor interactions, then the team may choose the

procedure on the left hand side of Figure 53. The op tions on the left side offer significant efficiencies, however, it should be noted that a recent meta-study of over 100 full factorial data sets suggests that in systems with five variables, one significant three-factor interactions is likely to be present and in systems with 10 variables, an average of five significant three-factor interactions are likely to be present, Li and Frey (2005). The noise strategies on the left of Figure 53 may save time and money but there have to be good reasons to believe that three-factor interactions are unlikely to arise.

Experience, judgment, and knowledge of engineering and science are critical formulating a noise strategy. These can be supplemented by effective processes guiding the work of a quality engineering team. It is hoped that the procedure proposed here (Figure 53) will be of value to practitioners seeking to implement robust design efficiently as part of their product development process.

5.5 Chapter Summary

Correlation among the noise factors has a much stronger effect on robust design if the Weak Hierarchy model is assumed than if the Strong Hierarchy model is assumed. Since the systems which follow strong hierarchy there is no significant three factor interactions. Impact of correlation among noise factors is pronounced when control-by-noise-by-noise interactions are present. Control-by-noise-by-noise interactions are present only in Weak Hierarchy Systems hence correlation has stronger effect on robust design is the system follows Weak Hierarchy.

The magnitude of the noise factors has a mild effect on the performance of robust design, except when the system has active three factor interactions and the correlation is moderate or high. In the presence of moderate or high correlation among noise, and active three factor interactions, control-by-noise and control-by-noise-by-noise interaction having same common control factor get confounded. This leads to sub-optimal selection of control factor settings.

In this chapter we first saw the impact of correlation and variance of induced noise factors on response surface instances generated via strong and weak hierarchy probability model. Next we ran similar noise strategies on six different case studies from various engineering domains to verify conclusions from hierarchical probability model. We saw that if system follows strong hierarchy then during robust design experiments we can neglect correlation and/or exaggerate the intensity of induced noise factors. We also designed an algorithm for implementing research findings in this chapter in practice to reduce amount of information needed to implement robust design methodology.

In the next chapter we will provide a brief summary and findings of all main chapters in this thesis. We will also present scope of future research in the area of reliability and robust design applied to complex systems.

Chapter 6: Conclusions and Future Work

6.1 Overview of research

Robust parameter design methods are used to make systems more reliable and robust to incoming variations in environmental effects, manufacturing processes and customer usage patterns. However, robust design can become expensive, time consuming, and/or resource intensive. Thus research that makes robust design less resource intensive and requires less number of experimental runs is of great value. Robust design methodology can be expressed as multi-response optimization problem. The objective functions of the problem being: maximizing reliability and robustness of systems, minimizing the information and/or resources required for robust design methodology, and minimizing the number of experimental runs needed.

Robust parameter design is an engineering methodology intended as a cost effective approach to improve the quality of products, processes and systems, Taguchi (1987), Robinson et al. (2004). Taguchi (1987) proposed that inputs to any system can be classified as control factors and noise factors. Control factors are those system parameters that can be easily controlled and manipulated. Noise factors are those system parameters that are difficult and/or costly to control and are presumed uncontrollable. Robust parameter design involves choosing optimal levels of the controllable factors in order to

obtain a target or optimal response with minimal variation. The challenge arises in obtaining optimal response due to the influence of the uncontrollable noise factors. Noise factors bring variability into the system, thus affecting the response. The aim is to properly choose the levels of control factors so that the process is robust or insensitive to the variation caused by noise factors.

Robust parameter design is among one of the most important developments in systems engineering in 20^{th} century, Clausing and Frey (2005). These methods seemed to have accounted for a significant part of quality differential that made Japanese manufacturing dominant during 1970s. Robust parameter design enables in smoother system integration, faster transition to production, and higher field reliability.

In chapter 1 we saw that robust parameter design process is crucial in improving the quality and reliability of any system/process. The main aim of this thesis is to find algorithms which will make robust design process a cost effective approach to implement. We explored noise strategies which make robust design processes require less number of experimental runs and less information about noise factor space to attain high quality improvement. We proposed random compound noise over compound noise since it does not require the knowledge of directionality of noise factors to be known. We also proposed TTBF strategy over compound noise for systems and processes in which factors effects were dense. TTBF strategy provides more resolution over compound noise hence is able to utilize all important interactions to improve the quality. We also proposed that

for the systems which have no active three-factor interactions correlation among noise factors can safely be neglected and/or intensity of noise factors can be exaggerated and still high robustness improvement will be achieved.

In chapter 2 the formulation of Hierarchical Probability Model was discussed. This will form the basis to compare different robust design methods statistically. First the regularities exhibited by engineering systems were discussed. Next those regularities were put in a mathematical format. The mathematical formulation would be used to generate response surface instances to analyze different robust design methods. We also discussed about selecting various parameters for Hierarchical Probability Model. We can have many variants of Hierarchical Probability Model. We discussed some of these variants.

In chapter 3 it was seen that Compound Noise as a robust design strategy is very effective on the systems which show *effect sparsity*. The reason for its effectiveness on sparse systems is, in compound noise all the noise factors are combined. Hence their individual impact on system's response is confounded. But if effects are sparse then the probability of the impact of two noise factors being oppositely confounded is extremely low. Hence compound noise is able to exploit all significant control-by-noise interactions for such systems, leading to its high effectiveness. We first ran two formulations of compound noise (simple and extreme) on response surface instances generated using strong and weak hierarchical probability model. This was done to confirm compound noise

effectiveness statistically. Next those formulations of compound noise were run on six different case studies from various engineering domains to verify conclusions from hierarchical probability model. In the end conditions for compound noise to be completely effective for both strong and weak hierarchy systems were outlined. We engineered an algorithm on the use of compound noise as a robust design method, based on our conclusions from response surface instances and case studies.

In chapter 4 it was found that TTBF Noise Factor strategy as a robust design strategy is very effective for all systems, even the ones which do not show *effect sparsity*. The reason for the effectiveness of TTBF strategy for all systems is that, it keeps the important noise factors in the system independent. Hence the individual impact of important noise factors on system's response is not confounded. TTBF strategy is able to exploit all significant control-by-noise interactions for such systems with very high probability, leading to its high effectiveness. We ran TTBF strategy on response surfaces generated using strong and weak hierarchical probability model. This was done to confirm its effectiveness statistically. Next TTBF Noise Factor strategy was run on six different case studies from various engineering domains to verify conclusions from hierarchical probability model. We also compared TTBF strategy with Compound Noise strategy for response surfaces generated using strong and weak hierarchical probability model. We also proposed a Hybrid Noise strategy with combines the effectiveness of both TTBF strategy and Compound Noise strategy. We devised an algorithm on the use

of TTBF strategy and Compound Noise strategy as robust design methods, based on our conclusions from response surface instances and case studies.

In chapter 5 the impact of correlation and variance of induced noise factors on response surface instances generated via strong and weak hierarchy probability model was seen. Next similar noise strategies were run on six different case studies from various engineering domains to verify conclusions from hierarchical probability model. We saw that if system follows strong hierarchy then during robust design experiments correlation can be neglected and/or the intensity of induced noise factors can be exaggerated. We also designed an algorithm for implementing research findings in this chapter in practice to reduce amount of information needed to implement robust design methodology.

6.2 Algorithms to improve quality of systems

In this section we will revisit two important flowcharts which define algorithms that should be used to deploy experimental runs efficiently in robust design practices. These algorithms also promise to reduce the amount of information required to improve the quality of systems/processes.

The results of this thesis can be used in an overall approach to deploying TTBF strategy and Compound Noise strategy as a robust design strategy. The flowchart in Figure 6.1 presents our suggestion for implementing these findings in practice. This flowchart is same as flow chart given in chapter 4, figure 4.6. First of all, practicing engineers must

define the scenario including what system is being improved, what objectives are being sought, and what design variables can be altered.

At this point, it may be possible to consider what assumptions can be made regarding *effect sparsity* for a given system/process. It should be noted here that we do not argue that engineers need to make a factual determination of *effect sparsity*. The experience on the system should be used to make decision. If engineers decide that effects are sparse, then they should follow the procedure on the left hand side of Figure 6.1. But if they decide that effects are dense, then they should figure out the most important noise factors for the system under consideration. In this case they should formulate TTBF strategy by keeping all the important noise factors independent in outer array and follow the procedure on the right hand side of Figure 6.1. Figure 6.1 will be of value to practitioners seeking to implement robust design efficiently and will reduce the amount of experimental runs required in order to improve the quality of a system/process.

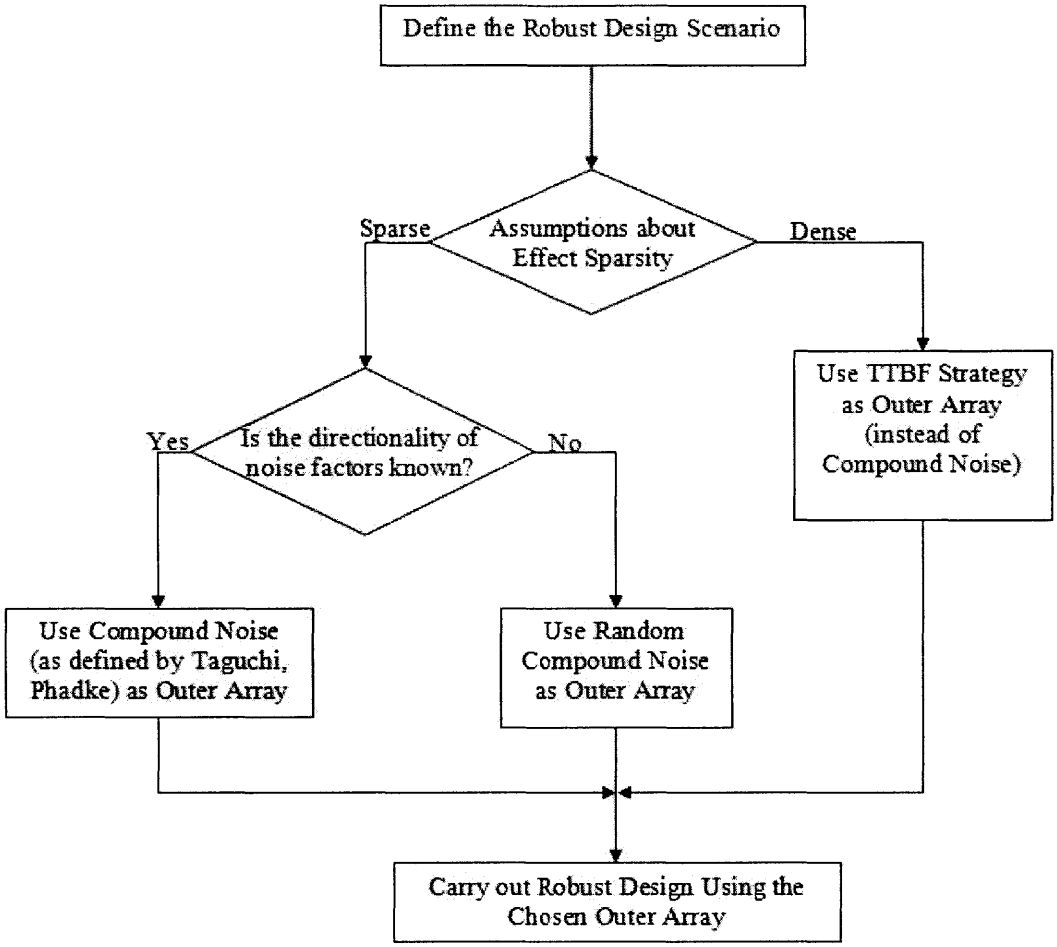


Figure 6.1: Suggested procedure for TTBF Strategy and Compound Noise in Robust Design

In order to minimize the information needed about noise factor space to run robust design process, flow chart in figure 6.2 presents our suggestions. First of all, the engineering team must define the scenario including what system is being improved, what objectives are being sought, and what design variables may be altered. After defining the

parameters of the scenario, it should be possible to assess the assumptions can be made regarding effect hierarchy. We are not suggesting that the team needs to make a factual determination of the existence of three-factor interactions. The experience of the team should be used to make an assessment which will necessarily be made under uncertainty. If the team decides that only weak hierarchy can be assumed, that is, that three-factor interactions might be present, then the team should choose the procedure on the right hand side of Figure 6.2. In this case, the team must decide whether it is possible or advisable to take measures that eliminate or reduce the possibility of three-factor interactions. In some cases, reformulation of the response, or the input variables, or redesign of the system can be effective. It is possible that formulating responses related to the energy transformations in the system will greatly reduce the likelihood of high order interactions. If the team finds it difficult to avoid the possibility of such interactions, the results in this chapter suggest that the noise strategy should seek to accurately represent both the intensity and correlations of the noises. If the type of system under consideration, the type of response, and the design variables are judged as unlikely to exhibit three-factor interactions, then the team may choose the procedure on the left hand side of Figure 6.2. The options on the left side offer significant efficiencies, however, it should be noted that a recent meta-study of over 100 full factorial data sets suggests that in systems with five variables, one significant three-factor interactions is likely to be present and in systems with 10 variables, an average of five significant three-factor interactions are likely to be present, Li and Frey (2005). The noise strategies on the left of Figure 6.2 may save time and money but there have to be good reasons to

believe that three-factor interactions are unlikely to arise. Experience, judgment, and knowledge of engineering and science are critical formulating a noise strategy. These can be supplemented by effective processes guiding the work of a quality engineering team. It is hoped that the procedure proposed here (Figure 6.2) will be of value to practitioners seeking to implement robust design efficiently as part of their product development process.

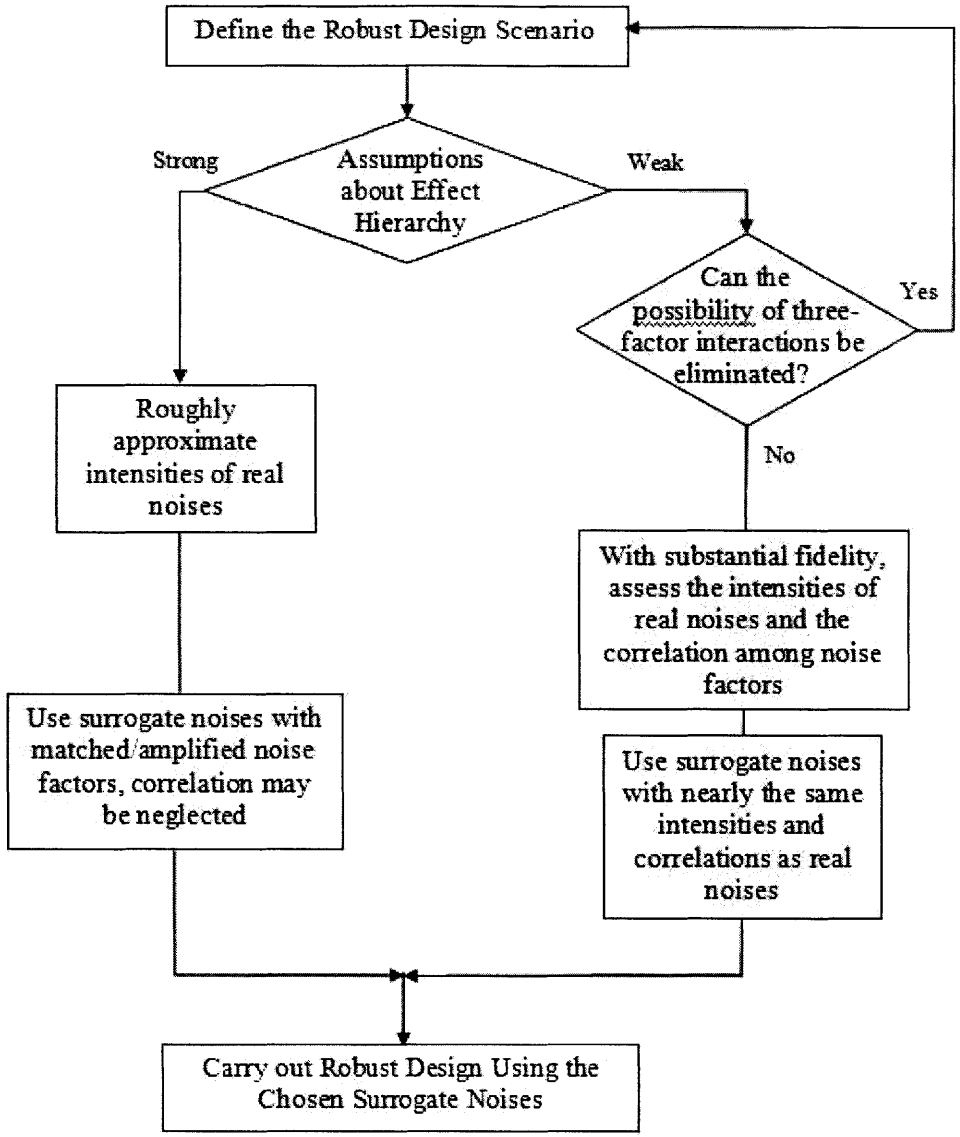


Figure 6.2: Flowchart to reduce information needed to implement Robust Design Methodology

6.3 Cost Benefit Analysis of Robust Design Methods

In this section we will present Cost-Benefit analysis of robust design methods discussed in this thesis. The results presented in this section are based on response surface instances generated using strong and weak hierarchy probability model and six case studies taken from various engineering domains. The case studies are Operational Amplifier (Op Amp), Phadke (1989), Passive Neuron Model (PNM), Tawfik and Durand (1994), Journal Bearing: Half Sommerfeld Solution, Hamrock, et al. (2004), Continuous-Stirred Tank Reactor (CSTR), Kalagnanam and Diwekar (1997), Temperature Control Circuit, Phadke (1989) and Slider Crank, Gao, et al. (1998).

We assume that a system contains **N** noise factors in its noise factor space and each of the noise factors is considered at two factor levels. Figure 6.3 shows the benefit that can be achieved for each of the robust design strategy out of the total maximum benefit that can be achieved for a given system. It orders the robust design methods in terms of increasing cost to implement them, from right to left. We try to characterize systems in figure 6.3, based on prior knowledge about noise factor space that an engineer has about systems. The percentages given in figure 6.3 are shows the amount of improvement that can be achieved for a given system statistically out of total possible improvement that could have been achieved using full factorial noise factor array. The * denotes that we will approximately achieve that much improvement for systems, if we run a given robust design process a number of times. The variance of response after running a given robust design method is shown at the bottom of the tree. With no improvement the spread of

response is wide, with 50% improvement the spread reduces and at optimal (or near optimal) setting of the system, spread of response is minimum.

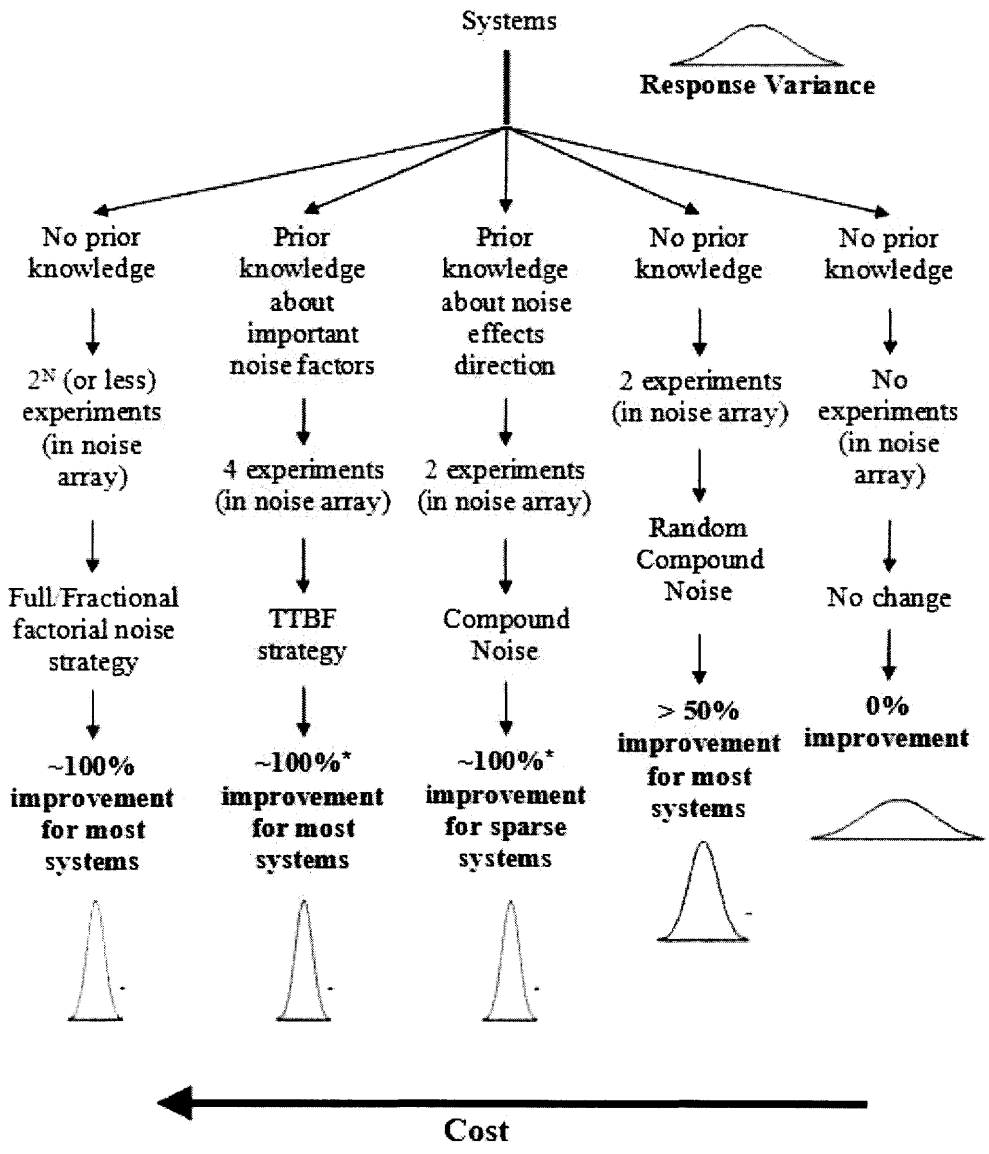


Figure 6. 3: Cost-Benefit Analysis of Robust Design Methods for reducing experimental runs

Figure 6.4 shows the benefit thatcan be achieved for each of the robust design strategy out of the total maximum benefit that can be achieved for a given system. It orders the robust design methods in terms of increasing cost to implement them, from right to left. We try to characterize systems in figure 6.4, based on highest order of active interaction present in the system, whether system follows strong hierarchy or weak hierarchy. We assume that noise factors to be correlated. We try to gage the error by neglecting correlation among noise factors and by amplifying the intensity of induced noise. The assumptions that require least amount of information are mentioned at extreme right of figure 6.4. As we go from right to left in figure 6.4 amount of information required about noise factor space of a given system increases. The percentages given in figure 6.4 are shows the amount of improvement that can be achieved for a given system statistically out of total possible improvement that could have been achieved using full factorial noise factor array at their actual intensities and including correlation among noise factors. The * denotes that we will approximately achieve that much improvement for systems, if a given robust design processis run a number of times.

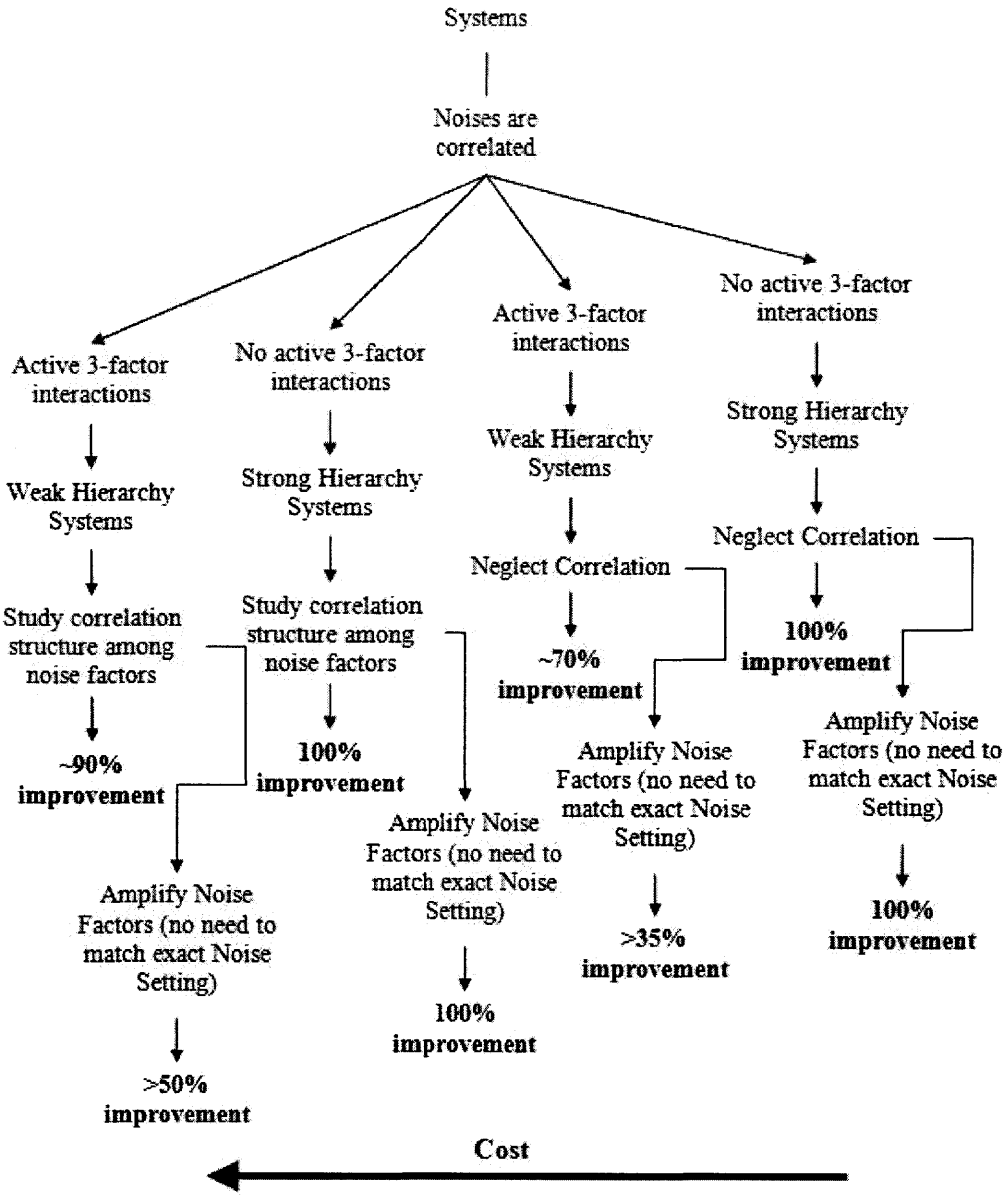


Figure 6. 4: Cost-Benefit Analysis of Robust Design Methods for minimizing information regarding noise factor space

6.4 Scope of future research

The algorithms presented in this thesis can be extremely helpful while designing new products and processes. These algorithms can be used to efficiently allocate resources among various sub-systems in a given complex system. Sub-systems which follow strong hierarchy would require fewer resources to improve quality as compared to sub-systems which follow weak hierarchy.

This would present a chance to incorporate robust design methods into product development cycle at very early stages. There can a platform study done in one of the design courses at MIT, where the improvement in quality that can be achieved in product by incorporating robust design methods at very early stages of product development cycle and allocating some resources for robust design methods can be gaged. We can document the improvement in quality for those products and can compare it with products where no such initiative was under taken.

Usually while working on legacy systems or prototypes, engineers learns a lot about the inherent properties of systems. This knowledge can be extremely helpful in devising specific robust design strategies for a particular class of systems which require less number of experimental run and less newer information about the actual system. These strategies can be even better than TTBF strategy and other noise strategies mentioned in this thesis. We can explore such noise strategies for each class of system, where deployment of robust design method is expensive affair.

REFERENCES

1. Allen, T. T., Bernshteyn, M., "Supersaturated Designs that maximize the probability of identifying active factors", Technometrics, Vol. 45, No. 1, 2003.

2. Box, G.E.P. and Meyer, R.D., "An Analysis for Unreplicated Fractional Factorials", Technometrics, 28, 11-18, 1986.

3. Chipman, H., Hamada, M. and Wu, C.F.J., "A Bayesian Variable-Selection Approach for Analyzing Designed Experiments with Complex Aliasing", Technometrics, 39, 372 381, 197.

4. Clausing, D., Frey, D., "Improving System Reliability by Failure-Mode Avoidance Including Four Concept Design Strategies", Systems Engineering, Vol. 8, No. 3, 2005.

5. Du, Xiaoping, Sudjianto, Agus, Chen, Wei, "An Integrated Framework for Probabilistic Optimization using Inverse Reliability Strategy", ASME, DETC, 2003, pp. 25-34.

6. Du, Xiaoping, Sudjianto, Agus, Chen, Wei, "An Integrated Framework for Optimization Under Uncertainty Using Inverse Reliability Strategy", Journal of Mechanical Design, July 2004.

7. Ejakov, Mikhail; Sudjianto, Agus; Pieprzak, John. Robustness and Performance Optimization of an IC Engine Using Computer Model and Surrogate Noise, ASME Design Automation Conference, Salt Lake City, Utah, USA, September 28 – October 2, 2004; DETC2004-57327.

8. Frey, D.D.; Li, X., "Validating robust parameter design methods", ASME Design Engineering Technical Conference 2004, Salt Lake City, Utah; DETC2004-57518.

9. Gao, J., Chase, K., Magleby, S., "Generalized 3-D tolerance analysis of mechanical assemblies with small kinematic adjustments", IIE Transactions, Vol. 30, No. 4, 1998.

10. Gigerenzer, G. and Goldstein, D., "Reasoning the Fast and Frugal Way: Models of Bounded Rationality", Psychological Review, 103(4), pp. 650-69, 1996.

11. Goldfarb, H.; Borror, C.; Montgomery, D. Mixture-process variable experiments with noise variables. Journal of Quality Technology, 35(4), 393-405, 2003.

12. Hamada, M. and Wu, C.F.J., "Analysis of Designed Experiments with Complex Aliasing", Journal of Quality Technology, 24, 130-137, 1992.

13. Hamrock, B., Schmid, S., Jacobson, B., "Fundamentals of Fluid Film Lubrication", 2nd edition, Marcel Dekker, 2004.

14. Heyden, Y. V., Kuttathatmmakul, S., Smeyers-Verbeke, J., Massart, D. L., "Supersaturated Designs for Robustness Testing", Analytical Chemistry, Vol. 72, No. 13, July, 2000.

15. Holcomb, D. R., Carlyle, W. M., "Some notes on the construction and evaluation of supersaturated designs", Quality and Reliability Engineering International, 18: 299-304, 2002.

16. Holcomb, D. R., Montgomery, D. C., Carlyle, W. M., "Analysis of supersaturated designs", Journal of Quality Technology, Vol. 35, No. 1, 2003.

17. Hou, X. Shirley, "On the use of compound noise factor in parameter design experiments", Applied Stochastic Models in Business and Industry, Vol. 18, pp. 225-243, 2002.

18. Joseph, V. R., and C. F. J. Wu, "Failure Amplification Method: An Information Maximization Approach to Categorical Response Optimization," Technometrics, Vol. 46, no. 1, pp. 1-12, 2004.

19. Kalagnanam, Jayant R., and Diwekar, Urmila, M., "An Efficient Sampling Technique for Off-line Quality Control", Technometrics, Vol. 39, No. 3, pp. 308-319, August, 1997.

20. Lenth, R., "Quick and Easy Analysis of Unreplicated Factorials", Technometrics, Vol. 31, No. 4, November, 1989.

21. Li, X., Frey, D., "Regularities in Data from Factorial Experiments", Complexity, 2005.

22. Phadke, Madhav S., "Quality Engineering Using Robust Design", Prentice Hall PTR, Englewood Cliffs, NJ, 1989.

23. Rao, S. S., "Reliability-Based Design", McGraw-Hill, Inc., 1992.

24. Robinson, T., Borror, C., Myers, R., "Robust Parameter Design: A Review", Quality and Reliability Engineering International, Vol. 20:81-101, 2004.

25. Satterthwaite, F. E., "Random balanced experimentation (with discussion)", Technometrics, 1(2): 111-137, 1959.

26. Siddall, J. N., "Probabilistic Engineering Design", Marcel Dekker, New York, 1983.

27. Taguchi, G., "System of Experimental Design: Engineering Methods to Optimize Quality and Minimize Costs", translated by Tung, L. W., Quality Resources: A Division of Kraus Organization Limited, White Plains, NY; and American Supplier Institute, Inc., Dearborn, MI, Vols. 1 and 2, 1987.

28. Twafik, B. and Durand, D., "Nonlinear Parameter Estimation by Linear Association: Application to a Five-Parameter Passive Neuron Model", IEEE Transactions on Biomedical Engineering, Vol. 41, No. 5, pp. 461-469, May, 1994.

29. Wu, C.F.J. and Hamada, M., "Experiments: Planning, Analysis, and Parameter Design Optimization", Wiley & Sons, Inc., NY, 2000.

Appendices: MATLAB® and Mathcad-11 Files

MATLAB® and Mathcad-11 files used in the thesis are given here.

1.1

```
% function compounding()

% We first assume the model parameters we want to use in RWH Strong Hierarchy
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Compounding is done by first finding the signs of b's for noise variables
% and low level of noise is set as the ones having signs opposite to that
% of their b-values and vice-versa for high setting. (INDEPENDENT NOISES).

% We will find optimal control factor setting for compounded noise using
% Transmitted Variance Model. And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and RWH
% model determines how active they are.

% WH for 200 models for Strong Hierarchy RWH Model

% 09/24/2004 by Jagmeet Singh

clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g'); % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d abcd');        % defining 2(5-1)(V) Array for NF's



modelpara=1;    % Defining which model parameters we would be using for 2nd order model
% Basic WH(1);  Basic low w(2); Basic 2nd order(3); Fitted WH(4);   Fitted
% low order(5); Fitted 2nd order(6)
modelparameter=models(modelpara);       % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1);  s1=modelparameter(1,2); s2=modelparameter(1,3);
w1=modelparameter(1,4); w2=modelparameter(1,5);   p=modelparameter(1,6);
p11=modelparameter(1,7); p01=modelparameter(1,8);   p00=modelparameter(1,9);% defining
parameters
```

```
ncf=7;  % # of CF's
nnf=5;  % # of NF's


counter_OCF_1 = 0;  % To increment when OCF from MC is same as from Noise Strategy 1
counter_OCF_c = 0;  % To increment when OCF from MC is same as from Compounded Noise

MU = [0    0     0     0     0];    % Defines the means of the Noise Variables been used
sigma_uncorrelated = eye(5);      % Define the covariance matrix for the assumed model
% the on diagonal elements are ones and rest all zero. EYE function
% generates Identity Matrix


maxnoisevar = 200;     % Maximum number of Noise Factor settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;         % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h1 = waitbar(0,'Running Models');
for modelcounter=1:maxmodels      % To run a given number of models
   [bi,bij]=RWH_2ndorder(ncf,nnf,c,s1,w1,p,p11,p01,p00);   % Finding beta values for a given
model

   nfsetting1 = [-1*sign(bi(1:5));
            sign(bi(1:5))];  % defining compounded noise based on b-values as described above
   nfsetting2 = [-1;
            1];            % defining low and high setting of compounded noise




% For 2(5-1)(V) Noise Array
   clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
   index_resp_matrix = 1;  % Counter for response matrix
   for cfruns = 1:128
      for nfruns = 1:16
         x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
         x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
         sumij=0; sumijk=0;
         for i=1:(nnf+ncf)
            for j=i+1:(nnf+ncf)
               sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
            end
         end
         y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
         ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
         index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
   end
```

```
clear NxC NxN CxCxN CxNxN;      % Clearing the History
% Fitting Response Model to 'y1' for 2(5-1)(V) Noise Array
nxc=1;            % Counter for Control by Noise Interactions
for nf=1:nnf      % Defining Control by Noise Interactions terms for Transmitted Variance
Model
   for cf=nnf+1:nnf+ncf
      NxC(:,nxc)=ResponseMatrix1(:,nf).*ResponseMatrix1(:,cf);
      nxc = nxc + 1;
   end
end
nxn=1;            % Counter for Noise by Noise Interactions
for nf1=1:nnf
   for nf2=nf1+1:nnf
      NxN(:,nxn)=ResponseMatrix1(:,nf1).*ResponseMatrix1(:,nf2);
      nxn = nxn + 1;
   end
end
cxnxn=1;          % Counter for Control X Noise X Noise Interaction
for nf1=1:nnf
   for nf2=nf1+1:nnf
      for cf = nnf+1:nnf+ncf

CxNxN(:,cxnxn)=ResponseMatrix1(:,cf).*ResponseMatrix1(:,nf1).*ResponseMatrix1(:,nf2);
         cxnxn = cxnxn + 1;
      end
   end
end
cxcxn=1;          % Counter for Control X Control X Noise Interaction
for nf=1:nnf
   for cf1=nnf+1:nnf+ncf
      for cf2=cf1+1:nnf+ncf

CxCxN(:,cxcxn)=ResponseMatrix1(:,cf1).*ResponseMatrix1(:,cf2).*ResponseMatrix1(:,nf);
         cxcxn = cxcxn + 1;
      end
   end
end

% To find the fitted model for Transmitted Variance Model
inputs = [ones(2048,1) ResponseMatrix1(:,1:12) NxC NxN CxNxN CxCxN];
[b,bint,r,rint,stats]=regress(ResponseMatrix1(:,13),inputs);
% b0(1) bi's(2:13)  CxN(14:48)  NxN(49:58)  CxNxN(59:128) CxCxN(129:233) The way b's
% are defined

% To Determine the variance under each CF setting for Transmitted
% Response Model

X = ff2n(7)*2 - 1;  % Defining x's for Response Model


% Defining B's for the ease
Bi(1:12) = b(2:13);
```

```
Bij(5,12) = 0;

index=14;      % CxN
for i=1:5
   for j=6:12
      Bij(i,j) = b(index);
      index=index+1;
   end
end

for i=1:5      % NxN
   for j=i+1:5
      Bij(i,j) = b(index);
      index = index+1;
   end
end

Bijk(12,12,12)=0;      % CxNxN
for i=1:5
   for j=i+1:5
      for k=6:12
         Bijk(i,j,k) = b(index);
         index = index+1;
      end
   end
end

for i=1:5      % CxCxN
   for j=6:12
      for k=j+1:12
         Bijk(i,j,k) = b(index);
         index=index+1;
      end
   end
end



% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:nnf      % First term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
      for j=6:12
         sum_b=sum_b+Bij(nf,j)*X(cf,j-5);
      end
      sum_c=0;
      for j=6:12
         for k=j+1:12
            sum_c=sum_c+Bijk(nf,j,k)*X(cf,j-5)*X(cf,k-5);
         end
```

```
        end
        sum1 = sum1 + (sum_a + sum_b +sum_c)^2;
      end

      for nf = 1:nnf     % Second term in Variance Model
        for j=nf+1:5
          sum_d=0;
          for k=6:12
            sum_d=sum_d+Bijk(nf,j,k)*X(cf,k-5);
          end
          sum2 = sum2 + (Bij(nf,j) + sum_d)^2;
        end
      end

      varianc(cf,1) = sum1 + sum2;
    end

    STDev_1 = varianc.^0.5;      % Stdev for each CF setting
    std_base_1 = STDev_1(1,1);
    op_std_1 = min(STDev_1);   % Finding least Stdev
    PredMin_1 = sortrows([STDev_1 X],1);
    OCF_N1 = PredMin_1(1,2:8);




    % For Compound Noise at 2 levels
    clear ResponseMatrix_c NxC NxN CxCxN CxNxN;  % Response Matrix = [NF CF Y]
    index_resp_matrix = 1;  % Counter for response matrix
    for cfruns = 1:128
      for nfruns = 1:2
        x(1,1:5)=nfsetting1(nfruns,:); % Defining Noise Factor Settings
        x(1,6:12)=cfsetting(cfruns,:);   % Defining CF Setting
        sumij=0; sumijk=0;
        for i=1:(nnf+ncf)
          for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
          end
        end
        y_c(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
        ResponseMatrix_c(index_resp_matrix,:)=[nfsetting2(nfruns,:) cfsetting(cfruns,:)
y_c(cfruns,nfruns)];
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end

    clear NxC NxN;     % Clearing the History
    % Fitting Response Model to 'y1' for 2 level Compound Noise

    nxc=1;            % Counter for Control by Noise Interactions
```

```
for nf=1:1      % Defining Control by Noise Interactions terms for Transmitted Variance Model
            % Since only one compounded noise factor
   for cf=2:8
      NxC(:,nxc)=ResponseMatrix_c(:,nf).*ResponseMatrix_c(:,cf);
      nxc = nxc + 1;
   end
end

cxcxn=1;          % Counter for Control X Control X Noise Interaction
for nf=1:1
   for cf1=2:1+ncf
      for cf2=cf1+1:1+ncf

CxCxN(:,cxcxn)=ResponseMatrix_c(:,cf1).*ResponseMatrix_c(:,cf2).*ResponseMatrix_c(:,nf);
         cxcxn = cxcxn + 1;
      end
   end
end

% To find the fitted model for Transmitted Variance Model
inputs = [ones(256,1) ResponseMatrix_c(:,1:8) NxC CxCxN];
[b,bint,r,rint,stats]=regress(ResponseMatrix_c(:,9),inputs);
% b0(1) bi's(2:9)  CxN(10:16) CxCxN(17:37) The way b's are defined

% To Determine the variance under each CF setting for Transmitted
% Response Model

X = ff2n(7)*2 - 1;  % Defining x's for Response Model


% Defining B's for the ease
Bi(1:8) = b(2:9);      % Main Effects
Bij(1:7) = b(10:16);   % C x Compounded_Noise

Bijk(8,8,8) = 0;
index = 17;
for i = 1:1          % CxC x Compounded_Noise
   for j = 2:8
      for k = j+1:8
         Bijk(i,j,k) = b(index);
         index = index+1;
      end
   end
end


% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:1      % First Term in Variance Model
      sum_a=Bi(nf);
```

```
        sum_b=0;
        for j = 1:7
           sum_b=sum_b + Bij(j)*X(cf,j);
        end
        sum_c=0;
        for j = 2:8
           for k = j+1:8
              sum_c = sum_c + Bijk(1,j,k)*X(cf,j-1)*X(cf,k-1);
           end
        end
        sum1 = sum1 + (sum_a + sum_b + sum_c)^2;
     end

     varianc(cf,1) = sum1+sum2;
  end


  STDev_c = varianc.^0.5;     % Stdev for each CF setting
  std_base_c = STDev_c(1,1);
  op_std_c = min(STDev_c);    % Finding Least STDev_1
  PredMin_c = sortrows([STDev_c X],1);
  OCF_c = PredMin_c(1,2:8);




  % Doing Monte Carlo for each setting of Control Factors
  clear ResponseMatrix_MC;    % Response Matrix = [CF Y's_for_CFsetting]
  [ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij, MU, sigma_uncorrelated);

  STDev_MC = varianc.^0.5;        % Stdev for each CF setting

  %std_base_MC = STDev_MC(1,1);
  std_base_MC = mean(STDev_MC);   % Base Stdev is taken as mean of all STDev's

  op_std_MC = min(STDev_MC);      % Finding least Stdev
  PredMin_MC = sortrows([STDev_MC X],1);
  OCF_MC = PredMin_MC(1,2:8);

  if OCF_MC == OCF_N1
     counter_OCF_1 = counter_OCF_1 + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and 2(5-1)(V) Noise Array
  end

  if OCF_MC == OCF_c
     counter_OCF_c = counter_OCF_c + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and Compounded Noise Factor
  end
```

```
   cf_1 = 0; cf_c = 0;    % To find number of Control factors whose settings are predicted
correctly

   for matching_cf = 1:7
      if OCF_N1(1,matching_cf) == OCF_MC(1,matching_cf)
         cf_1 = cf_1 + 1;
      end
      if OCF_c(1,matching_cf) == OCF_MC(1,matching_cf)
         cf_c = cf_c + 1;
      end
   end

   matching_noise1(modelcounter) = cf_1;  % To store # of OCF Matched
   matching_compound(modelcounter) = cf_c;    % To store # of OCF Matched



   % Determining the Optimal Standard Deviation from Monte Carlo
   Opt_MC = std_for_cfsetting(ResponseMatrix_MC, OCF_MC);

   % Determining the Optimal Standard Deviation from Noise Strategy 1
   Opt_1 = std_for_cfsetting(ResponseMatrix_MC, OCF_N1);

   % Determining the Optimal Standard Deviation from Compound Noise
   Opt_c = std_for_cfsetting(ResponseMatrix_MC, OCF_c);


   std_base = std_base_MC;    % Base Stdev is taken as mean of all STDev's


   % Storing and Analysing Results
   std_fraction1(modelcounter) = (Opt_1 / Opt_MC);
   std_fraction2(modelcounter) = (Opt_c / Opt_MC);

   % Storing Improvement Ratios for Noise Strategy 1 and Compound Noise
   std_fraction3(modelcounter) = ((std_base - Opt_1)/(std_base - Opt_MC + 1e-10));
   std_fraction4(modelcounter) = ((std_base - Opt_c)/(std_base - Opt_MC + 1e-10));

   Y(modelcounter) = (std_base - Opt_c)/std_base;
   X1(modelcounter) = (std_base - Opt_MC)/std_base;

   waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d',modelcounter))
end
close(h1);  % Close waitbar

% saving workspace
save variables;

output;
```

1.2

```
function vector=models(modelpara)
% It defines the parameters that we would be using for Relaxed-Weak
% Heredity model
% Reference Chipman, Hamada and Wu (1997) and Li and Frey (2005)
% 03/04/2004 by Jagmeet Singh


Table1 = [10   1    1     1      1
      10   1    1     0.1    0.1
      10   1    0     1      1
      15   1/3  2/3   1      1
      15   1/3  2/3   0.1    0.1
      15   1/3  0     1      1];

Table2 = [0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0        0     0     0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0        0     0     0];

Table1= [Table1 Table2];      % for input to Main Model
vector=Table1(modelpara,:);
```

1.3

```
function [bi,bij]=RWH_2ndorder(ncf,nnf,c,s1,w1,p,p11,p01,p00)

% Developing Strong Hierarchy RWH Model (without ERROR)
% INPUTS: # of CF's,   # of NF's,    c,    s1,    w1,    p,
% p00,    p01,    p11

% OUTPUT: bi's and bij's
% Developed on 03/03/2004 by Jagmeet Singh

for i=1:(ncf+nnf)  % Defining t as mentioned in the writeup
   if i <= nnf
      t(i)=w1;
   else
      t(i)=1;
   end
end

delta=unifrnd(0,1,[1 nnf+ncf]);    % Defining delta
for i=1:nnf+ncf                    % Prob (delta_i = 1) = p
   if delta(1,i) <= p
      delta(1,i)=1;
   else
      delta(1,i)=0;
   end
end

deltaij(1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
   for j=i+1:(nnf+ncf)
      sum_deltas=delta(1,i)+delta(1,j);  % Finding the sum of delta-i + delta-j
      deltaij(i,j)=unifrnd(0,1);         % Defining delta-ij [0,1]

      if sum_deltas == 0                 % Defining delta-ij when both main factors are inactive
         if deltaij(i,j) <= p00
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 1                 % Defining delta-ij when one of the factors is active
         if deltaij(i,j) <= p01
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 2                 % Defining delta-ij when both the factors are active
```

```
        if deltaij(i,j) <= p11
           deltaij(i,j)=1;
        else
           deltaij(i,j)=0;
        end
     end

  end
end

for i=1:nnf+ncf                    % Defining bi's for the CF's and NF's
  if delta(1,i) == 0
     bi(1,i)=t(i)*normrnd(0,1);
  else
     bi(1,i)=t(i)*normrnd(0,c);
  end
end

bij(1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
  for j=i+1:(nnf+ncf)
     if deltaij(i,j) == 0
        bij(i,j)=t(i)*t(j)*normrnd(0,s1);
     else
        bij(i,j)=t(i)*t(j)*normrnd(0,c*s1);
     end
  end
end
```

1.4

```
% Function to find the variance of the response once it is given the
% Control Factor (CF) setting and the Matrix with contains the response and
% cfsetting

% Inputs: Response Matrix and Required CF setting

% Output: Standard Deviation of Response for the given CF setting

% 03/24/2004 by Jagmeet Singh

function [std_dev] = std_for_cfsetting(ResponseMatrix, setting)

global cfsetting w2 ncf nnf maxnoisevar;

clear ysetting;

for i = 1:128
   if setting == ResponseMatrix(i,1:7)
      ysetting = ResponseMatrix(i,8:maxnoisevar+7);
      std_dev = std(ysetting);
   end
end
```

1.5

```
% Function Var_cf_setting takes inputs from RWH model, mean of Noises and
% the Covariance Matrix, which shows that Noise variables are independent.
% It then generates the Response for 200 Noise factors settings and finds
% the variance at each control factor setting for full model

% 03/16/2004 by Jagmeet Singh

function [ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij, MU, sigma)

global cfsetting w2 ncf nnf maxnoisevar;

X = ff2n(7)*2 - 1;  % Defining x's for Response Model
nfsetting = lhsnorm(MU, sigma, maxnoisevar);
for cfruns = 1:size(X,1)
   for nfruns = 1:maxnoisevar
      x(1,1:5)=nfsetting(nfruns,:);   % Defining Noise Factor Settings
      x(1,6:12)=X(cfruns,:);  % Defining CF Settings
      sumij=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end
      y_MC(cfruns,nfruns)=bi*x' + sumij + normrnd(0,w2);
   end
end

ResponseMatrix_MC = [X y_MC];   % Storing CF setting and Yassumed for that setting
varianc = (var(y_MC'))';   % Finding the Variance for each CF setting
```

1.6

```
% Function to plot the outputs of the response

% 03/16/2004 by Jagmeet Singh

function output()

load variables;

% Plotting and Analysing Results from the runs

% Pie Chart when OCF_MC is same as OCF_N1
figure; success=[(counter_OCF_1) (maxmodels-counter_OCF_1)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Same CF levels: ';'Diff CF levels: '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in prediction of OCF_M_C from OCF_n_o_i_s_e _s_t_r_a_t_e_g_y _1
for',num2str(maxmodels),' response surface instances']);
hgsave('pie1');

% Pie Chart when OCF_MC is same as OCF_c
figure; success=[(counter_OCF_c) (maxmodels-counter_OCF_c)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Same CF levels: ';'Diff CF levels: '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in prediction of OCF_M_C from OCF_e_x_t_r_e_m_e_c_o_m_p_o_u_n_d
_n_o_i_s_e  for',num2str(maxmodels),' response surface instances']);
hgsave('pie2');


% Plotting Histograms
figure;
hist(std_fraction1);
title(['Histogram for Fraction of OPT.STD_n_o_i_s_e_1 to OPT.STD_M_C for
',num2str(maxmodels),' response surface instances']);
colormap cool; iq = prctile(std_fraction1,[25 50 75]); tmax = max(hist(std_fraction1));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('OPT.STD.N1 / OPT.STD.MC');
ylabel(' number of instances of response surface ');
hgsave('fig1');

figure;
```

```
hist(std_fraction2);
title(['Histogram for Fraction of OPT.STD_c_o_m_p_o_u_n_d to OPT.STD_M_C for
',num2str(maxmodels),' response surface instances']);
colormap cool; iq = prctile(std_fraction2,[25 50 75]); tmax = max(hist(std_fraction2));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('OPT.STD.Compd / OPT.STD.MC');
ylabel(' number of instances of response surface ');
hgsave('fig2');

figure;
hist(std_fraction3);
title(['Histogram for Fraction of (STD_b_a_s_e - OPT.STD_n_o_i_s_e_1)/(STD_b_a_s_e -
OPT.STD_M_C) for ',num2str(maxmodels),' response surface instances']);
colormap cool; iq = prctile(std_fraction3,[25 50 75]); tmax = max(hist(std_fraction3));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Base-OPT.STD.N1)/(STD.Base-OPT.STD.MC)');
ylabel(' number of instances of response surface ');
hgsave('fig3');

figure;
hist(std_fraction4);
title(['Histogram for Fraction of (STD_b_a_s_e - OPT.STD_c_o_m_p_o_u_n_d)/(STD_b_a_s_e -
OPT.STD_M_C) for ',num2str(maxmodels),' response surface instances']);
colormap cool; iq = prctile(std_fraction4,[25 50 75]); tmax = max(hist(std_fraction4));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Base-OPT.STD.Compd)/(STD.Base-OPT.STD.MC)');
ylabel(' number of instances of response surface ');
hgsave('fig4');

figure;
plot(X1,Y,'*','color','r');
X2 = [ones(size(X1')) X1'];
a = X2\Y';
Y2 = a'*X2';
B = [X1' Y2'];
i0 = regress(Y',X1');
B = sortrows(B,1);
hold on;
line([B(1,1);B(maxmodels,1)], [ B(1,2);B(maxmodels,2)],'Color','g', 'LineWidth', 0.5);
line([0;B(maxmodels,1)],[0;i0*B(maxmodels,1)],'LineWidth',1.5);
title(['For 2^n^d Order Response Surfaces : Plotting (STD_b_a_s_e -
STD_c_o_m_p_d)/STD_b_a_s_e vs (STD_b_a_s_e - STD_o_p_t)/STD_b_a_s_e and slope = ',
num2str(a(2,1))]);
xlabel('(STD_b_a_s_e - STD_o_p_t)/STD_b_a_s_e');
ylabel('(STD_b_a_s_e - STD_c_o_m_p_d)/STD_b_a_s_e');
hgsave('fig5');
```

```
prob_pos = 0;   % To find the probability that compounding will yield positive improvement
for index = 1:maxmodels
   if Y(1,index) >= 0.00
      prob_pos = prob_pos + 1;
   end
end

% Printing the results
sprintf(['mean(Number of OCF matched for Noise Strategy 1) =
',num2str(mean(matching_noise1))])
sprintf(['mean(Number of OCF matched for Compounded Noise) =
',num2str(mean(matching_compound))])
sprintf([' Probability that Compounding will Yield Positive Improvement =
',num2str(prob_pos/maxmodels)])
```

2.1

```
% function compounding()

% We first assume the model parameters we want to use in RWH Strong Hierarchy
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Compounding is done by first finding the signs of b's for noise variables
% and low level of noise is set as the ones having signs opposite to that
% of their b-values and vice-versa for high setting. (INDEPENDENT NOISES).

% We will find optimal control factor setting for compounded noise using
% Transmitted Variance Model. And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and RWH
% model determines how active they are.

% WH for 200 models for Strong Hierarchy RWH Model

% 09/24/2004 by Jagmeet Singh

clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g'); % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d abcd');      % defining 2(5-1)(V) Array for NF's



modelpara=1;   % Defining which model parameters we would be using for Strong Hierarchy
model
% Basic WH(1); Basic low w(2); Basic 2nd order(3); Fitted WH(4);   Fitted
% low order(5); Fitted 2nd order(6)
modelparameter=models(modelpara);      % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1); s1=modelparameter(1,2); s2=modelparameter(1,3);
w1=modelparameter(1,4); w2=modelparameter(1,5);   p=modelparameter(1,6);
p11=modelparameter(1,7); p01=modelparameter(1,8);   p00=modelparameter(1,9);% defining
parameters

ncf=7;  % # of CF's
nnf=5;  % # of NF's


counter_OCF_1 = 0;  % To increment when OCF from MC is same as from Noise Strategy 1
```

```
counter_OCF_c = 0;  % To increment when OCF from MC is same as from Compounded Noise

MU = [0    0    0    0    0];   % Defines the means of the Noise Variables been used
sigma_uncorrelated = eye(5);    % Define the covariance matrix for the assumed model
% the on diagonal elements are ones and rest all zero. EYE function
% generates Identity Matrix


maxnoisevar = 200;   % Maximum number of Noise Factor settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;       % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h1 = waitbar(0,'Running Models');
for modelcounter=1:maxmodels      % To run a given number of models
  [bi,bij]=RWH_2ndorder(ncf,nnf,c,s1,w1,p,p11,p01,p00);   % Finding beta values for a given
model

  nfsetting1 = [-1 -1 -1 -1 -1;
              1  1  1  1  1];  % defining simple compound noise
  nfsetting2 = [-1;
             1];           % defining low and high setting of compounded noise




% For 2(5-1)(V) Noise Array
  clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
  index_resp_matrix = 1;  % Counter for response matrix
  for cfruns = 1:128
    for nfruns = 1:16
      x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
      sumij=0; sumijk=0;
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
        end
      end
      y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
      ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
      index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
    end
  end

  clear NxC NxN CxCxN CxNxN;      % Clearing the History
  % Fitting Response Model to 'y1' for 2(5-1)(V) Noise Array
  nxc=1;            % Counter for Control by Noise Interactions
```

```
  for nf=1:nnf      % Defining Control by Noise Interactions terms for Transmitted Variance
Model
     for cf=nnf+1:nnf+ncf
        NxC(:,nxc)=ResponseMatrix1(:,nf).*ResponseMatrix1(:,cf);
        nxc = nxc + 1;
     end
  end
  nxn=1;          % Counter for Noise by Noise Interactions
  for nf1=1:nnf
     for nf2=nf1+1:nnf
        NxN(:,nxn)=ResponseMatrix1(:,nf1).*ResponseMatrix1(:,nf2);
        nxn = nxn + 1;
     end
  end
  cxnxn=1;          % Counter for Control X Noise X Noise Interaction
  for nf1=1:nnf
     for nf2=nf1+1:nnf
        for cf = nnf+1:nnf+ncf

CxNxN(:,cxnxn)=ResponseMatrix1(:,cf).*ResponseMatrix1(:,nf1).*ResponseMatrix1(:,nf2);
           cxnxn = cxnxn + 1;
        end
     end
  end
  cxcxn=1;          % Counter for Control X Control X Noise Interaction
  for nf=1:nnf
     for cf1=nnf+1:nnf+ncf
        for cf2=cf1+1:nnf+ncf

CxCxN(:,cxcxn)=ResponseMatrix1(:,cf1).*ResponseMatrix1(:,cf2).*ResponseMatrix1(:,nf);
           cxcxn = cxcxn + 1;
        end
     end
  end

  % To find the fitted model for Transmitted Variance Model
  inputs = [ones(2048,1) ResponseMatrix1(:,1:12) NxC NxN CxNxN CxCxN];
  [b,bint,r,rint,stats]=regress(ResponseMatrix1(:,13),inputs);
  % b0(1) bi's(2:13)  CxN(14:48)  NxN(49:58)  CxNxN(59:128) CxCxN(129:233) The way b's
  % are defined

  % To Determine the variance under each CF setting for Transmitted
  % Response Model

  X = ff2n(7)*2 - 1;  % Defining x's for Response Model


  % Defining B's for the ease
  Bi(1:12) = b(2:13);
  Bij(5,12) = 0;

  index=14;      % CxN
```

```
for i=1:5
   for j=6:12
      Bij(i,j) = b(index);
      index=index+1;
   end
end

for i=1:5      % NxN
   for j=i+1:5
      Bij(i,j) = b(index);
      index = index+1;
   end
end

Bijk(12,12,12)=0;      % CxNxN
for i=1:5
   for j=i+1:5
      for k=6:12
         Bijk(i,j,k) = b(index);
         index = index+1;
      end
   end
end

for i=1:5     % CxCxN
   for j=6:12
      for k=j+1:12
         Bijk(i,j,k) = b(index);
         index=index+1;
      end
   end
end



% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:nnf      % First term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
      for j=6:12
         sum_b=sum_b+Bij(nf,j)*X(cf,j-5);
      end
      sum_c=0;
      for j=6:12
         for k=j+1:12
            sum_c=sum_c+Bijk(nf,j,k)*X(cf,j-5)*X(cf,k-5);
         end
      end
      sum1 = sum1 + (sum_a + sum_b +sum_c)^2;
   end
```

```
    for nf = 1:nnf      % Second term in Variance Model
       for j=nf+1:5
          sum_d=0;
          for k=6:12
             sum_d=sum_d+Bijk(nf,j,k)*X(cf,k-5);
          end
          sum2 = sum2 + (Bij(nf,j) + sum_d)^2;
       end
    end

    varianc(cf,1) = sum1 + sum2;
  end

  STDev_1 = varianc.^0.5;      % Stdev for each CF setting
  std_base_1 = STDev_1(1,1);
  op_std_1 = min(STDev_1);    % Finding least Stdev
  PredMin_1 = sortrows([STDev_1 X],1);
  OCF_N1 = PredMin_1(1,2:8);




  % For Compound Noise at 2 levels
  clear ResponseMatrix_c NxC NxN CxCxN CxNxN;  % Response Matrix = [NF CF Y]
  index_resp_matrix = 1;  % Counter for response matrix
  for cfruns = 1:128
     for nfruns = 1:2
        x(1,1:5)=nfsetting1(nfruns,:); % Defining Noise Factor Settings
        x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
        sumij=0; sumijk=0;
        for i=1:(nnf+ncf)
           for j=i+1:(nnf+ncf)
              sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
           end
        end
        y_c(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
        ResponseMatrix_c(index_resp_matrix,:)=[nfsetting2(nfruns,:) cfsetting(cfruns,:)
y_c(cfruns,nfruns)];
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
     end
  end

  clear NxC NxN;      % Clearing the History
  % Fitting Response Model to 'y1' for 2 level Compound Noise

  nxc=1;              % Counter for Control by Noise Interactions

  for nf=1:1      % Defining Control by Noise Interactions terms for Transmitted Variance Model
                  % Since only one compounded noise factor
```

```
    for cf=2:8
      NxC(:,nxc)=ResponseMatrix_c(:,nf).*ResponseMatrix_c(:,cf);
      nxc = nxc + 1;
    end
  end


  cxcxn=1;          % Counter for Control X Control X Noise Interaction
  for nf=1:1
    for cf1=2:1+ncf
      for cf2=cf1+1:1+ncf

CxCxN(:,cxcxn)=ResponseMatrix_c(:,cf1).*ResponseMatrix_c(:,cf2).*ResponseMatrix_c(:,nf);
        cxcxn = cxcxn + 1;
      end
    end
  end

  % To find the fitted model for Transmitted Variance Model
  inputs = [ones(256,1) ResponseMatrix_c(:,1:8) NxC CxCxN];
  [b,bint,r,rint,stats]=regress(ResponseMatrix_c(:,9),inputs);
  % b0(1) bi's(2:9)  CxN(10:16) CxCxN(17:37) The way b's are defined

  % To Determine the variance under each CF setting for Transmitted
  % Response Model

  X = ff2n(7)*2 - 1;  % Defining x's for Response Model


  % Defining B's for the ease
  Bi(1:8) = b(2:9);      % Main Effects
  Bij(1:7) = b(10:16);   % C x Compounded_Noise

  Bijk(8,8,8) = 0;
  index = 17;
  for i = 1:1          % CxC x Compounded_Noise
    for j = 2:8
      for k = j+1:8
        Bijk(i,j,k) = b(index);
        index = index+1;
      end
    end
  end


  % Fitting Transmitted Variance Model
  for cf = 1:128
    sum1=0;sum2=0;
    for nf = 1:1       % First Term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
```

```
        for j = 1:7
            sum_b=sum_b + Bij(j)*X(cf,j);
        end
        sum_c=0;
        for j = 2:8
            for k = j+1:8
                sum_c = sum_c + Bijk(1,j,k)*X(cf,j-1)*X(cf,k-1);
            end
        end
        sum1 = sum1 + (sum_a + sum_b + sum_c)^2;
    end

    varianc(cf,1) = sum1+sum2;
end


STDev_c = varianc.^0.5;     % Stdev for each CF setting
std_base_c = STDev_c(1,1);
op_std_c = min(STDev_c);    % Finding Least STDev_1
PredMin_c = sortrows([STDev_c X],1);
OCF_c = PredMin_c(1,2:8);




% Doing Monte Carlo for each setting of Control Factors
clear ResponseMatrix_MC;    % Response Matrix = [CF Y's_for_CFsetting]
[ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij, MU, sigma_uncorrelated);

STDev_MC = varianc.^0.5;        % Stdev for each CF setting

%std_base_MC = STDev_MC(1,1);
std_base_MC = mean(STDev_MC);   % Base Stdev is taken as mean of all STDev's

op_std_MC = min(STDev_MC);      % Finding least Stdev
PredMin_MC = sortrows([STDev_MC X],1);
OCF_MC = PredMin_MC(1,2:8);

if OCF_MC == OCF_N1
    counter_OCF_1 = counter_OCF_1 + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and 2(5-1)(V) Noise Array
end

if OCF_MC == OCF_c
    counter_OCF_c = counter_OCF_c + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and Compounded Noise Factor
end
```

```
   cf_1 = 0; cf_c = 0;     % To find number of Control factors whose settings are predicted
correctly

   for matching_cf = 1:7
      if OCF_N1(1,matching_cf) == OCF_MC(1,matching_cf)
         cf_1 = cf_1 + 1;
      end
      if OCF_c(1,matching_cf) == OCF_MC(1,matching_cf)
         cf_c = cf_c + 1;
      end
   end

   matching_noise1(modelcounter) = cf_1;   % To store # of OCF Matched
   matching_compound(modelcounter) = cf_c;     % To store # of OCF Matched



   % Determining the Optimal Standard Deviation from Monte Carlo
   Opt_MC = std_for_cfsetting(ResponseMatrix_MC, OCF_MC);

   % Determining the Optimal Standard Deviation from Noise Strategy 1
   Opt_1 = std_for_cfsetting(ResponseMatrix_MC, OCF_N1);

   % Determining the Optimal Standard Deviation from Compound Noise
   Opt_c = std_for_cfsetting(ResponseMatrix_MC, OCF_c);


   std_base = std_base_MC;     % Base Stdev is taken as mean of all STDev's


   % Storing and Analysing Results
   std_fraction1(modelcounter) = (Opt_1 / Opt_MC);
   std_fraction2(modelcounter) = (Opt_c / Opt_MC);

   % Storing Improvement Ratios for Noise Strategy 1 and Compound Noise
   std_fraction3(modelcounter) = ((std_base - Opt_1)/(std_base - Opt_MC + 1e-10));
   std_fraction4(modelcounter) = ((std_base - Opt_c)/(std_base - Opt_MC + 1e-10));

   Y(modelcounter) = (std_base - Opt_c)/std_base;
   X1(modelcounter) = (std_base - Opt_MC)/std_base;

   waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d',modelcounter))
end
close(h1);  % Close waitbar

% saving workspace
save variables;

output;
```

3.1

```
% function compounding()

% We first assume the model parameters we want to use in RWH Weak Hierarchy
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Compounding is done by first finding the signs of b's for noise variables
% and low level of noise is set as the ones having signs opposite to that
% of their b-values and vice-versa for high setting. (INDEPENDENT NOISES).

% We will find optimal control factor setting for compounded noise using
% Transmitted Variance Model. And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and RWH
% model determines how active they are.

% WH for 200 models for Weak Hierarchy RWH Model

% 09/24/2004 by Jagmeet Singh

clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g'); % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d abcd');      % defining 2(5-1)(V) Array for NF's



modelpara=1;   % Defining which model parameters we would be using for Weak Hierarchy
model
% Basic WH(1); Basic low w(2); Basic 2nd order(3); Fitted WH(4);  Fitted
% low order(5); Fitted 2nd order(6)
modelparameter=models(modelpara);      % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1);    s1=modelparameter(1,2);    s2=modelparameter(1,3);
w1=modelparameter(1,4);   w2=modelparameter(1,5);    p=modelparameter(1,6);
p11=modelparameter(1,7);  p01=modelparameter(1,8);   p00=modelparameter(1,9);
p111=modelparameter(1,10); p011=modelparameter(1,11); p001=modelparameter(1,12);
p000=modelparameter(1,13);% defining parameters

ncf=7;  % # of CF's
nnf=5;  % # of NF's
```

```
counter_OCF_1 = 0;  % To increment when OCF from MC is same as from Noise Strategy 1
counter_OCF_c = 0;  % To increment when OCF from MC is same as from Compounded Noise

MU = [0    0     0     0     0];   % Defines the means of the Noise Variables been used
sigma_uncorrelated = eye(5);     % Define the covariance matrix for the assumed model
% the on diagonal elements are ones and rest all zero. EYE function
% generates Identity Matrix


maxnoisevar = 200;    % Maximum number of Noise Factor settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;        % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h1 = waitbar(0,'Running Models');
for modelcounter=1:maxmodels      % To run a given number of models
  [bi,bij,bijk]=RWH_3rdorder(ncf,nnf,c,s1,s2,w1,p,p11,p01,p00,p111,p011,p001,p000);   %
Finding beta values for a given model

  nfsetting1 = [-1*sign(bi(1:5));
          sign(bi(1:5))];  % defining compounded noise based on b-values as described above
  nfsetting2 = [-1;
          1];          % defining low and high setting of compounded noise


% For 2(5-1)(V) Noise Array
  clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
  index_resp_matrix = 1;  % Counter for response matrix
  for cfruns = 1:128
    for nfruns = 1:16
      x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);   % Defining CF Setting
      sumij=0; sumijk=0;
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
        end
      end
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          for k=j+1:(nnf+ncf)
            sumijk=sumijk+bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
          end
        end
      end
      y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
```

```
        ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end

    clear NxC NxN CxCxN CxNxN;      % Clearing the History
    % Fitting Response Model to 'y1' for 2(5-1)(V) Noise Array
    nxc=1;            % Counter for Control by Noise Interactions
    for nf=1:nnf      % Defining Control by Noise Interactions terms for Transmitted Variance
Model
      for cf=nnf+1:nnf+ncf
        NxC(:,nxc)=ResponseMatrix1(:,nf).*ResponseMatrix1(:,cf);
        nxc = nxc + 1;
      end
    end
    nxn=1;            % Counter for Noise by Noise Interactions
    for nf1=1:nnf
      for nf2=nf1+1:nnf
        NxN(:,nxn)=ResponseMatrix1(:,nf1).*ResponseMatrix1(:,nf2);
        nxn = nxn + 1;
      end
    end
    cxnxn=1;          % Counter for Control X Noise X Noise Interaction
    for nf1=1:nnf
      for nf2=nf1+1:nnf
        for cf = nnf+1:nnf+ncf

CxNxN(:,cxnxn)=ResponseMatrix1(:,cf).*ResponseMatrix1(:,nf1).*ResponseMatrix1(:,nf2);
          cxnxn = cxnxn + 1;
        end
      end
    end
    cxcxn=1;          % Counter for Control X Control X Noise Interaction
    for nf=1:nnf
      for cf1=nnf+1:nnf+ncf
        for cf2=cf1+1:nnf+ncf

CxCxN(:,cxcxn)=ResponseMatrix1(:,cf1).*ResponseMatrix1(:,cf2).*ResponseMatrix1(:,nf);
          cxcxn = cxcxn + 1;
        end
      end
    end

    % To find the fitted model for Transmitted Variance Model
    inputs = [ones(2048,1) ResponseMatrix1(:,1:12) NxC NxN CxNxN CxCxN];
    [b,bint,r,rint,stats]=regress(ResponseMatrix1(:,13),inputs);
    % b0(1) bi's(2:13)  CxN(14:48)  NxN(49:58)  CxNxN(59:128) CxCxN(129:233) The way b's
    % are defined

    % To Determine the variance under each CF setting for Transmitted
    % Response Model
```

```
X = ff2n(7)*2 - 1;  % Defining x's for Response Model


% Defining B's for the ease
Bi(1:12) = b(2:13);
Bij(5,12) = 0;

index=14;      % CxN
for i=1:5
   for j=6:12
      Bij(i,j) = b(index);
      index=index+1;
   end
end

for i=1:5      % NxN
   for j=i+1:5
      Bij(i,j) = b(index);
      index = index+1;
   end
end

Bijk(12,12,12)=0;      % CxNxN
for i=1:5
   for j=i+1:5
      for k=6:12
         Bijk(i,j,k) = b(index);
         index = index+1;
      end
   end
end

for i=1:5     % CxCxN
   for j=6:12
      for k=j+1:12
         Bijk(i,j,k) = b(index);
         index=index+1;
      end
   end
end



% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:nnf     % First term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
      for j=6:12
```

```
        sum_b=sum_b+Bij(nf,j)*X(cf,j-5);
      end
      sum_c=0;
      for j=6:12
        for k=j+1:12
          sum_c=sum_c+Bijk(nf,j,k)*X(cf,j-5)*X(cf,k-5);
        end
      end
      sum1 = sum1 + (sum_a + sum_b +sum_c)^2;
    end

    for nf = 1:nnf      % Second term in Variance Model
      for j=nf+1:5
        sum_d=0;
        for k=6:12
          sum_d=sum_d+Bijk(nf,j,k)*X(cf,k-5);
        end
        sum2 = sum2 + (Bij(nf,j) + sum_d)^2;
      end
    end

    varianc(cf,1) = sum1 + sum2;
end

STDev_1 = varianc.^0.5;       % Stdev for each CF setting
std_base_1 = STDev_1(1,1);
op_std_1 = min(STDev_1);    % Finding least Stdev
PredMin_1 = sortrows([STDev_1 X],1);
OCF_N1 = PredMin_1(1,2:8);




% For Compound Noise at 2 levels
clear ResponseMatrix_c;  % Response Matrix = [NF CF Y]
index_resp_matrix = 1;  % Counter for response matrix
for cfruns = 1:128
   for nfruns = 1:2
      x(1,1:5)=nfsetting1(nfruns,:); % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
      sumij=0; sumijk=0;
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
        end
      end
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          for k=j+1:(nnf+ncf)
            sumijk=sumijk+bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
```

```
            end
          end
        end
        y_c(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
        ResponseMatrix_c(index_resp_matrix,:)=[nfsetting2(nfruns,:) cfsetting(cfruns,:)
y_c(cfruns,nfruns)];
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end

    clear NxC NxN CxNxN CxCxN;      % Clearing the History
    % Fitting Response Model to 'y1' for 2 level Compound Noise

    nxc=1;            % Counter for Control by Noise Interactions

    for nf=1:1      % Defining Control by Noise Interactions terms for Transmitted Variance Model
                % Since only one compounded noise factor
      for cf=2:8
        NxC(:,nxc)=ResponseMatrix_c(:,nf).*ResponseMatrix_c(:,cf);
        nxc = nxc + 1;
      end
    end


    cxcxn=1;          % Counter for Control X Control X Noise Interaction
    for nf=1:1
      for cf1=2:1+ncf
        for cf2=cf1+1:1+ncf

CxCxN(:,cxcxn)=ResponseMatrix_c(:,cf1).*ResponseMatrix_c(:,cf2).*ResponseMatrix_c(:,nf);
          cxcxn = cxcxn + 1;
        end
      end
    end

    % To find the fitted model for Transmitted Variance Model
    inputs = [ones(256,1) ResponseMatrix_c(:,1:8) NxC CxCxN];
    [b,bint,r,rint,stats]=regress(ResponseMatrix_c(:,9),inputs);
    % b0(1) bi's(2:9)  CxN(10:16) CxCxN(17:37) The way b's are defined

    % To Determine the variance under each CF setting for Transmitted
    % Response Model

    X = ff2n(7)*2 - 1;  % Defining x's for Response Model


    % Defining B's for the ease
    Bi(1:8) = b(2:9);      % Main Effects
    Bij(1:7) = b(10:16);    % C x Compounded_Noise

    Bijk(8,8,8) = 0;
```

```
index = 17;
for i = 1:1          % CxC x Compounded_Noise
   for j = 2:8
      for k = j+1:8
         Bijk(i,j,k) = b(index);
         index = index+1;
      end
   end
end


% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:1      % First Term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
      for j = 1:7
         sum_b=sum_b + Bij(j)*X(cf,j);
      end
      sum_c=0;
      for j = 2:8
         for k = j+1:8
            sum_c = sum_c + Bijk(1,j,k)*X(cf,j-1)*X(cf,k-1);
         end
      end
      sum1 = sum1 + (sum_a + sum_b + sum_c)^2;
   end

   varianc(cf,1) = sum1+sum2;
end

STDev_c = varianc.^0.5;    % Stdev for each CF setting
std_base_c = STDev_c(1,1);
op_std_c = min(STDev_c);   % Finding Least STDev_1
PredMin_c = sortrows([STDev_c X],1);
OCF_c = PredMin_c(1,2:8);




% Doing Monte Carlo for each setting of Control Factors
clear ResponseMatrix_MC;   % Response Matrix = [CF Y's_for_CFsetting]
[ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij,bijk, MU, sigma_uncorrelated);

STDev_MC = varianc.^0.5;       % Stdev for each CF setting

%std_base_MC = STDev_MC(1,1);
std_base_MC = mean(STDev_MC);  % Base Stdev is taken as mean of all STDev's
```

```
op_std_MC = min(STDev_MC);      % Finding least Stdev
PredMin_MC = sortrows([STDev_MC X],1);
OCF_MC = PredMin_MC(1,2:8);

if OCF_MC == OCF_N1
   counter_OCF_1 = counter_OCF_1 + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and 2(5-1)(V) Noise Array
end

if OCF_MC == OCF_c
   counter_OCF_c = counter_OCF_c + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and Compounded Noise Factor
end



cf_1 = 0; cf_c = 0;     % To find number of Control factors whose settings are predicted
correctly

for matching_cf = 1:7
   if OCF_N1(1,matching_cf) == OCF_MC(1,matching_cf)
      cf_1 = cf_1 + 1;
   end
   if OCF_c(1,matching_cf) == OCF_MC(1,matching_cf)
      cf_c = cf_c + 1;
   end
end

matching_noise1(modelcounter) = cf_1;   % To store # of OCF Matched
matching_compound(modelcounter) = cf_c;     % To store # of OCF Matched



% Determining the Optimal Standard Deviation from Monte Carlo
Opt_MC = std_for_cfsetting(ResponseMatrix_MC, OCF_MC);

% Determining the Optimal Standard Deviation from Noise Strategy 1
Opt_1 = std_for_cfsetting(ResponseMatrix_MC, OCF_N1);

% Determining the Optimal Standard Deviation from Compound Noise
Opt_c = std_for_cfsetting(ResponseMatrix_MC, OCF_c);


std_base = std_base_MC;     % Base Stdev is taken as mean of all STDev's


% Storing and Analysing Results
std_fraction1(modelcounter) = (Opt_1 / Opt_MC);
std_fraction2(modelcounter) = (Opt_c / Opt_MC);
```

```
   % Storing Improvement Ratios for Noise Strategy 1 and Compound Noise
   std_fraction3(modelcounter) = ((std_base - Opt_1)/(std_base - Opt_MC + 1e-10));
   std_fraction4(modelcounter) = ((std_base - Opt_c)/(std_base - Opt_MC + 1e-10));

   Y(modelcounter) = (std_base - Opt_c)/std_base;
   X1(modelcounter) = (std_base - Opt_MC)/std_base;

   waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d',modelcounter))
end
close(h1);  % Close waitbar

% saving workspace
save variables;

output;
```

3.2

```
function vector=models(modelpara)
% It defines the parameters that we would be using for Relaxed-Weak
% Heredity model
% Reference Chipman, Hamada and Wu (1997) and Li and Frey (2005)
% 03/04/2004 by Jagmeet Singh


Table1 = [10   1    1     1      1
      10   1    1     0.1    0.1
      10   1    0     1      1
      15   1/3  2/3   1      1
      15   1/3  2/3   0.1    0.1
      15   1/3  0     1      1];

Table2 = [0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0        0     0     0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0        0     0     0];

Table1= [Table1 Table2];      % for input to Main Model
vector=Table1(modelpara,:);
```

3.3

```
function [bi,bij,bijk]=RWH_3rdorder(ncf,nnf,c,s1,s2,w1,p,p11,p01,p00,p111,p011,p001,p000)

% Developing Weak Hierarchy RWH Model (without ERROR) Including the Demand and
% Capacity noises for Phase 5 study.

% INPUTS: # of CF's,   # of NF's,    c,    s1,    s2,    w1,    p,
% p00,    p01,      p11,         p111,  p011,  p001   p000

% OUTPUT: bi's, bij's,and bijk's
% Developed on 03/24/2004 by Jagmeet Singh

% Defining the intensity of Noise wrt range of Control Factor setting (w1)
w1 = 1.0;

for i=1:(ncf+nnf)  % Defining t as mentioned in the writeup
   if i <= nnf
      t(i)=w1;
   else
      t(i)=1;
   end
end



delta=unifrnd(0,1,[1 nnf+ncf]);    % Defining delta
for i=1:nnf+ncf                    % Prob (delta_i = 1) = p
   if delta(1,i) <= p
      delta(1,i)=1;
   else
      delta(1,i)=0;
   end
end

deltaij(1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
   for j=i+1:(nnf+ncf)
      sum_deltas=delta(1,i)+delta(1,j);  % Finding the sum of delta-i + delta-j
      deltaij(i,j)=unifrnd(0,1);         % Defining delta-ij [0,1]

      if sum_deltas == 0                 % Defining delta-ij when both main factors are inactive
         if deltaij(i,j) <= p00
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 1                 % Defining delta-ij when one of the factors is active
```

```
        if deltaij(i,j) <= p01
           deltaij(i,j)=1;
        else
           deltaij(i,j)=0;
        end
     end

     if sum_deltas == 2          % Defining delta-ij when both the factors are active
        if deltaij(i,j) <= p11
           deltaij(i,j)=1;
        else
           deltaij(i,j)=0;
        end
     end

  end
end

% Defining delta-ijk
deltaijk(1:(nnf+ncf),1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
  for j=i+1:(nnf+ncf)
     for k=j+1:(nnf+ncf)
        sum_deltas=delta(1,i)+delta(1,j)+delta(1,k);   % Finding the sum of delta's
        deltaijk(i,j,k)=unifrnd(0,1);          % Defining delta-ijk [0,1]

        if sum_deltas == 0          % Defining delta-ijk when all 3 main effects are inactive
           if deltaijk(i,j,k) <= p000
              deltaijk(i,j,k)=1;
           else
              deltaijk(i,j,k)=0;
           end
        end

        if sum_deltas == 1          % Defining delta-ijk when all 2 main effects are inactive
           if deltaijk(i,j,k) <= p001
              deltaijk(i,j,k)=1;
           else
              deltaijk(i,j,k)=0;
           end
        end

        if sum_deltas == 2          % Defining delta-ijk when all 2 main effects are active
           if deltaijk(i,j,k) <= p011
              deltaijk(i,j,k)=1;
           else
              deltaijk(i,j,k)=0;
           end
        end

        if sum_deltas == 3          % Defining delta-ijk when all 3 main effects are active
           if deltaijk(i,j,k) <= p111
```

```
            deltaijk(i,j,k)=1;
          else
            deltaijk(i,j,k)=0;
          end
        end

    end
  end
end


for i=1:nnf+ncf                    % Defining bi's for the CF's and NF's
  if delta(1,i) == 0
    bi(1,i)=t(i)*normrnd(0,1);
  else
    bi(1,i)=t(i)*normrnd(0,c);
  end
end

bij(1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
  for j=i+1:(nnf+ncf)
    if deltaij(i,j) == 0
      bij(i,j)=t(i)*t(j)*normrnd(0,s1);
    else
      bij(i,j)=t(i)*t(j)*normrnd(0,c*s1);
    end
  end
end

bijk(1:nnf+ncf,1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
  for j=i+1:nnf+ncf
    for k=j+1:nnf+ncf
      if deltaijk(i,j,k) == 0
        bijk(i,j,k)=t(i)*t(j)*t(k)*normrnd(0,s2);
      else
        bijk(i,j,k)=t(i)*t(j)*t(k)*normrnd(0,c*s2);
      end
    end
  end
end
```

3.4

```
% Function to find the variance of the response once it is given the
% Control Factor setting and the Matrix with contains the response and
% cfsetting

% Inputs: Response Matrix and Required CF setting

% Output: Standard Deviation of Response for the given CF setting

% 03/24/2004 by Jagmeet Singh

function [std_dev] = std_for_cfsetting(ResponseMatrix, setting)

global cfsetting w2 ncf nnf maxnoisevar;

clear ysetting;

for i = 1:128
   if setting == ResponseMatrix(i,1:7)
      ysetting = ResponseMatrix(i,8:maxnoisevar+7);
      std_dev = std(ysetting);
   end
end
```

3.5

```
% Function Var_cf_setting takes inputs from RWH model, mean of Noises and
% the Covariance Matrix, which shows that Noise variables are independent.
% It then generates the Response for 200 Noise factors settings and finds
% the variance at each control factor setting for full model

% 08/10/2004 by Jagmeet Singh

function [ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij, bijk, MU, sigma)

global cfsetting w2 ncf nnf maxnoisevar;

X = ff2n(7)*2 - 1;  % Defining x's for Response Model
nfsetting = lhsnorm(MU, sigma, maxnoisevar);
for cfruns = 1:size(X,1)
   for nfruns = 1:maxnoisevar
      x(1,1:5)=nfsetting(nfruns,:);   % Defining Noise Factor Settings
      x(1,6:12)=X(cfruns,:);  % Defining CF Settings
      sumij=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end
      sumijk=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            for k=j+1:(nnf+ncf)
               sumijk=sumijk + bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
            end
         end
      end
      y_MC(cfruns,nfruns)=bi*x' + sumij + sumijk + normrnd(0,w2);
   end
end

ResponseMatrix_MC = [X y_MC];   % Storing CF setting and Yassumed for that setting
varianc = (var(y_MC'))';   % Finding the Variance for each CF setting
```

3.6

```
% Function to plot the outputs of the response

% 03/16/2004 by Jagmeet Singh

function output()

load variables;

% Plotting and Analysing Results from the runs

% Pie Chart when OCF_MC is same as OCF_N1
figure; success=[(counter_OCF_1) (maxmodels-counter_OCF_1)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Same CF levels: ';'Diff CF levels: '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in prediction of OCF_M_C from OCF_n_o_i_s_e _s_t_r_a_t_e_g_y _1
for',num2str(maxmodels),' response surface instances']);
hgsave('pie1');

% Pie Chart when OCF_MC is same as OCF_c
figure; success=[(counter_OCF_c) (maxmodels-counter_OCF_c)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Same CF levels: ';'Diff CF levels: '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in prediction of OCF_M_C from OCF_e_x_t_r_e_m_e_c_o_m_p_o_u_n_d
_n_o_i_s_e  for',num2str(maxmodels),' response surface instances']);
hgsave('pie2');


% Plotting Histograms
figure;
hist(std_fraction1);
title(['Histogram for Fraction of OPT.STD_n_o_i_s_e_1 to OPT.STD_M_C for
',num2str(maxmodels),' response surface instances']);
colormap cool; iq = prctile(std_fraction1,[25 50 75]); tmax = max(hist(std_fraction1));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('OPT.STD.N1 / OPT.STD.MC');
ylabel(' number of instances of response surface ');
hgsave('fig1');

figure;
```

```
hist(std_fraction2);
title(['Histogram for Fraction of OPT.STD_c_o_m_p_o_u_n_d to OPT.STD_M_C for
',num2str(maxmodels),' response surface instances']);
colormap cool; iq = prctile(std_fraction2,[25 50 75]); tmax = max(hist(std_fraction2));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('OPT.STD.Compd / OPT.STD.MC');
ylabel(' number of instances of response surface ');
hgsave('fig2');

figure;
hist(std_fraction3);
title(['Histogram for Fraction of (STD_b_a_s_e - OPT.STD_n_o_i_s_e_1)/(STD_b_a_s_e -
OPT.STD_M_C) for ',num2str(maxmodels),' response surface instances']);
colormap cool; iq = prctile(std_fraction3,[25 50 75]); tmax = max(hist(std_fraction3));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Base-OPT.STD.N1)/(STD.Base-OPT.STD.MC)');
ylabel(' number of instances of response surface ');
hgsave('fig3');

figure;
hist(std_fraction4);
title(['Histogram for Fraction of (STD_b_a_s_e - OPT.STD_c_o_m_p_o_u_n_d)/(STD_b_a_s_e -
OPT.STD_M_C) for ',num2str(maxmodels),' response surface instances']);
colormap cool; iq = prctile(std_fraction4,[25 50 75]); tmax = max(hist(std_fraction4));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Base-OPT.STD.Compd)/(STD.Base-OPT.STD.MC)');
ylabel(' number of instances of response surface ');
hgsave('fig4');

figure;
plot(X1,Y,'*','color','r');
X2 = [ones(size(X1')) X1'];
a = X2\Y';
Y2 = a'*X2';
B = [X1' Y2'];
i0 = regress(Y',X1');
B = sortrows(B,1);
hold on;
line([B(1,1);B(maxmodels,1)], [ B(1,2);B(maxmodels,2)],'Color','g', 'LineWidth', 0.5);
line([0;B(maxmodels,1)],[0;i0*B(maxmodels,1)],'LineWidth',1.5);
title(['For 2^n^d Order Response Surfaces : Plotting (STD_b_a_s_e -
STD_c_o_m_p_d)/STD_b_a_s_e vs (STD_b_a_s_e - STD_o_p_t)/STD_b_a_s_e and slope = ',
num2str(a(2,1))]);
xlabel('(STD_b_a_s_e - STD_o_p_t)/STD_b_a_s_e');
ylabel('(STD_b_a_s_e - STD_c_o_m_p_d)/STD_b_a_s_e');
hgsave('fig5');
```

```
prob_pos = 0;   % To find the probability that compounding will yield positive improvement
for index = 1:maxmodels
   if Y(1,index) >= 0.00
      prob_pos = prob_pos + 1;
   end
end

% Printing the results
sprintf(['mean(Number of OCF matched for Noise Strategy 1) =
',num2str(mean(matching_noise1))])
sprintf(['mean(Number of OCF matched for Compounded Noise) =
',num2str(mean(matching_compound))])
sprintf([' Probability that Compounding will Yield Positive Improvement =
',num2str(prob_pos/maxmodels)])
```

4.1

```
% function compounding()

% We first assume the model parameters we want to use in RWH Weak Hierarchy
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Compounding is done by first finding the signs of b's for noise variables
% and low level of noise is set as the ones having signs opposite to that
% of their b-values and vice-versa for high setting. (INDEPENDENT NOISES).

% We will find optimal control factor setting for compounded noise using
% Transmitted Variance Model. And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and RWH
% model determines how active they are.

% WH for 200 models for Weak Hierarchy RWH Model

% 09/24/2004 by Jagmeet Singh

clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g'); % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d abcd');      % defining 2(5-1)(V) Array for NF's



modelpara=1;   % Defining which model parameters we would be using for 3rd order model
% Basic WH(1); Basic low w(2); Basic 2nd order(3); Fitted WH(4);  Fitted
% low order(5); Fitted 2nd order(6)
modelparameter=models(modelpara);      % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1);    s1=modelparameter(1,2);    s2=modelparameter(1,3);
w1=modelparameter(1,4);    w2=modelparameter(1,5);    p=modelparameter(1,6);
p11=modelparameter(1,7);   p01=modelparameter(1,8);   p00=modelparameter(1,9);
p111=modelparameter(1,10); p011=modelparameter(1,11); p001=modelparameter(1,12);
p000=modelparameter(1,13);% defining parameters

ncf=7; % # of CF's
nnf=5; % # of NF's


counter_OCF_1 = 0; % To increment when OCF from MC is same as from Noise Strategy 1
```

```
counter_OCF_c = 0;  % To increment when OCF from MC is same as from Compounded Noise

MU = [0    0     0     0     0];   % Defines the means of the Noise Variables been used
sigma_uncorrelated = eye(5);     % Define the covariance matrix for the assumed model
% the on diagonal elements are ones and rest all zero. EYE function
% generates Identity Matrix


maxnoisevar = 200;    % Maximum number of Noise Factor settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;        % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h1 = waitbar(0,'Running Models');
for modelcounter=1:maxmodels      % To run a given number of models
  [bi,bij,bijk]=RWH_3rdorder(ncf,nnf,c,s1,s2,w1,p,p11,p01,p00,p111,p011,p001,p000);   %
Finding beta values for a given model

  nfsetting1 = [-1 -1 -1 -1 -1;
          1  1  1  1  1];  % defining simple compounded noise
  nfsetting2 = [-1;
          1];          % defining low and high setting of compounded noise




% For 2(5-1)(V) Noise Array
  clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
  index_resp_matrix = 1;  % Counter for response matrix
  for cfruns = 1:128
    for nfruns = 1:16
      x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
      sumij=0; sumijk=0;
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
        end
      end
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          for k=j+1:(nnf+ncf)
            sumijk=sumijk+bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
          end
        end
      end
      y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
      ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
```

```
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end

    clear NxC NxN CxCxN CxNxN;      % Clearing the History
    % Fitting Response Model to 'y1' for 2(5-1)(V) Noise Array
    nxc=1;              % Counter for Control by Noise Interactions
    for nf=1:nnf        % Defining Control by Noise Interactions terms for Transmitted Variance
Model
      for cf=nnf+1:nnf+ncf
        NxC(:,nxc)=ResponseMatrix1(:,nf).*ResponseMatrix1(:,cf);
        nxc = nxc + 1;
      end
    end
    nxn=1;              % Counter for Noise by Noise Interactions
    for nf1=1:nnf
      for nf2=nf1+1:nnf
        NxN(:,nxn)=ResponseMatrix1(:,nf1).*ResponseMatrix1(:,nf2);
        nxn = nxn + 1;
      end
    end
    cxnxn=1;            % Counter for Control X Noise X Noise Interaction
    for nf1=1:nnf
      for nf2=nf1+1:nnf
        for cf = nnf+1:nnf+ncf

CxNxN(:,cxnxn)=ResponseMatrix1(:,cf).*ResponseMatrix1(:,nf1).*ResponseMatrix1(:,nf2);
          cxnxn = cxnxn + 1;
        end
      end
    end
    cxcxn=1;            % Counter for Control X Control X Noise Interaction
    for nf=1:nnf
      for cf1=nnf+1:nnf+ncf
        for cf2=cf1+1:nnf+ncf

CxCxN(:,cxcxn)=ResponseMatrix1(:,cf1).*ResponseMatrix1(:,cf2).*ResponseMatrix1(:,nf);
          cxcxn = cxcxn + 1;
        end
      end
    end

    % To find the fitted model for Transmitted Variance Model
    inputs = [ones(2048,1) ResponseMatrix1(:,1:12) NxC NxN CxNxN CxCxN];
    [b,bint,r,rint,stats]=regress(ResponseMatrix1(:,13),inputs);
    % b0(1) bi's(2:13)  CxN(14:48)  NxN(49:58)  CxNxN(59:128) CxCxN(129:233) The way b's
    % are defined

    % To Determine the variance under each CF setting for Transmitted
    % Response Model

    X = ff2n(7)*2 - 1;  % Defining x's for Response Model
```

```
% Defining B's for the ease
Bi(1:12) = b(2:13);
Bij(5,12) = 0;

index=14;      % CxN
for i=1:5
   for j=6:12
      Bij(i,j) = b(index);
      index=index+1;
   end
end

for i=1:5      % NxN
   for j=i+1:5
      Bij(i,j) = b(index);
      index = index+1;
   end
end

Bijk(12,12,12)=0;      % CxNxN
for i=1:5
   for j=i+1:5
      for k=6:12
         Bijk(i,j,k) = b(index);
         index = index+1;
      end
   end
end

for i=1:5     % CxCxN
   for j=6:12
      for k=j+1:12
         Bijk(i,j,k) = b(index);
         index=index+1;
      end
   end
end



% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:nnf     % First term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
      for j=6:12
         sum_b=sum_b+Bij(nf,j)*X(cf,j-5);
      end
```

```
      sum_c=0;
      for j=6:12
         for k=j+1:12
            sum_c=sum_c+Bijk(nf,j,k)*X(cf,j-5)*X(cf,k-5);
         end
      end
      sum1 = sum1 + (sum_a + sum_b +sum_c)^2;
   end

   for nf = 1:nnf     % Second term in Variance Model
      for j=nf+1:5
         sum_d=0;
         for k=6:12
            sum_d=sum_d+Bijk(nf,j,k)*X(cf,k-5);
         end
         sum2 = sum2 + (Bij(nf,j) + sum_d)^2;
      end
   end

   varianc(cf,1) = sum1 + sum2;
end

STDev_1 = varianc.^0.5;      % Stdev for each CF setting
std_base_1 = STDev_1(1,1);
op_std_1 = min(STDev_1);   % Finding least Stdev
PredMin_1 = sortrows([STDev_1 X],1);
OCF_N1 = PredMin_1(1,2:8);




% For Compound Noise at 2 levels
clear ResponseMatrix_c;  % Response Matrix = [NF CF Y]
index_resp_matrix = 1;  % Counter for response matrix
for cfruns = 1:128
   for nfruns = 1:2
      x(1,1:5)=nfsetting1(nfruns,:); % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);   % Defining CF Setting
      sumij=0; sumijk=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            for k=j+1:(nnf+ncf)
               sumijk=sumijk+bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
            end
```

```
            end
        end
        y_c(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
        ResponseMatrix_c(index_resp_matrix,:)=[nfsetting2(nfruns,:) cfsetting(cfruns,:)
y_c(cfruns,nfruns)];
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end

    clear NxC NxN CxNxN CxCxN;      % Clearing the History
    % Fitting Response Model to 'y1' for 2 level Compound Noise

    nxc=1;            % Counter for Control by Noise Interactions

    for nf=1:1      % Defining Control by Noise Interactions terms for Transmitted Variance Model
                % Since only one compounded noise factor
      for cf=2:8
        NxC(:,nxc)=ResponseMatrix_c(:,nf).*ResponseMatrix_c(:,cf);
        nxc = nxc + 1;
      end
    end


    cxcxn=1;          % Counter for Control X Control X Noise Interaction
    for nf=1:1
      for cf1=2:1+ncf
        for cf2=cf1+1:1+ncf

CxCxN(:,cxcxn)=ResponseMatrix_c(:,cf1).*ResponseMatrix_c(:,cf2).*ResponseMatrix_c(:,nf);
          cxcxn = cxcxn + 1;
        end
      end
    end

    % To find the fitted model for Transmitted Variance Model
    inputs = [ones(256,1) ResponseMatrix_c(:,1:8) NxC CxCxN];
    [b,bint,r,rint,stats]=regress(ResponseMatrix_c(:,9),inputs);
    % b0(1) bi's(2:9)  CxN(10:16) CxCxN(17:37) The way b's are defined

    % To Determine the variance under each CF setting for Transmitted
    % Response Model

    X = ff2n(7)*2 - 1;  % Defining x's for Response Model


    % Defining B's for the ease
    Bi(1:8) = b(2:9);      % Main Effects
    Bij(1:7) = b(10:16);    % C x Compounded_Noise

    Bijk(8,8,8) = 0;
    index = 17;
```

```
for i = 1:1          % CxC x Compounded_Noise
   for j = 2:8
      for k = j+1:8
         Bijk(i,j,k) = b(index);
         index = index+1;
      end
   end
end


% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:1      % First Term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
      for j = 1:7
         sum_b=sum_b + Bij(j)*X(cf,j);
      end
      sum_c=0;
      for j = 2:8
         for k = j+1:8
            sum_c = sum_c + Bijk(1,j,k)*X(cf,j-1)*X(cf,k-1);
         end
      end
      sum1 = sum1 + (sum_a + sum_b + sum_c)^2;
   end

   varianc(cf,1) = sum1+sum2;
end

STDev_c = varianc.^0.5;    % Stdev for each CF setting
std_base_c = STDev_c(1,1);
op_std_c = min(STDev_c);   % Finding Least STDev_1
PredMin_c = sortrows([STDev_c X],1);
OCF_c = PredMin_c(1,2:8);




% Doing Monte Carlo for each setting of Control Factors
clear ResponseMatrix_MC;   % Response Matrix = [CF Y's_for_CFsetting]
[ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij,bijk, MU, sigma_uncorrelated);

STDev_MC = varianc.^0.5;       % Stdev for each CF setting

%std_base_MC = STDev_MC(1,1);
std_base_MC = mean(STDev_MC);  % Base Stdev is taken as mean of all STDev's
```

```
op_std_MC = min(STDev_MC);      % Finding least Stdev
PredMin_MC = sortrows([STDev_MC X],1);
OCF_MC = PredMin_MC(1,2:8);

if OCF_MC == OCF_N1
   counter_OCF_1 = counter_OCF_1 + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and 2(5-1)(V) Noise Array
end

if OCF_MC == OCF_c
   counter_OCF_c = counter_OCF_c + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and Compounded Noise Factor
end



cf_1 = 0; cf_c = 0;     % To find number of Control factors whose settings are predicted
correctly

for matching_cf = 1:7
   if OCF_N1(1,matching_cf) == OCF_MC(1,matching_cf)
      cf_1 = cf_1 + 1;
   end
   if OCF_c(1,matching_cf) == OCF_MC(1,matching_cf)
      cf_c = cf_c + 1;
   end
end

matching_noise1(modelcounter) = cf_1;   % To store # of OCF Matched
matching_compound(modelcounter) = cf_c;     % To store # of OCF Matched



% Determining the Optimal Standard Deviation from Monte Carlo
Opt_MC = std_for_cfsetting(ResponseMatrix_MC, OCF_MC);

% Determining the Optimal Standard Deviation from Noise Strategy 1
Opt_1 = std_for_cfsetting(ResponseMatrix_MC, OCF_N1);

% Determining the Optimal Standard Deviation from Compound Noise
Opt_c = std_for_cfsetting(ResponseMatrix_MC, OCF_c);


std_base = std_base_MC;     % Base Stdev is taken as mean of all STDev's


% Storing and Analysing Results
std_fraction1(modelcounter) = (Opt_1 / Opt_MC);
std_fraction2(modelcounter) = (Opt_c / Opt_MC);
```

```
   % Storing Improvement Ratios for Noise Strategy 1 and Compound Noise
   std_fraction3(modelcounter) = ((std_base - Opt_1)/(std_base - Opt_MC + 1e-10));
   std_fraction4(modelcounter) = ((std_base - Opt_c)/(std_base - Opt_MC + 1e-10));

   Y(modelcounter) = (std_base - Opt_c)/std_base;
   X1(modelcounter) = (std_base - Opt_MC)/std_base;

   waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d',modelcounter))
end
close(h1);  % Close waitbar

% saving workspace
save variables;

output;
```

5.1

```
% We first assume the model parameters we want to use in RWH Strong Hierarchy
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Compounding is done by first finding the signs of b's for noise variables
% and low level of noise is set as the ones having signs opposite to that
% of their b-values and vice-versa for high setting. (INDEPENDENT NOISES).

% We will find optimal control factor setting for compounded noise.
% And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and RWH
% model determines how active they are.

% Find Improvement Ratio for each value of p11, p01, p00 from 0.01 to 1.00.
% The p (prob. of active main effects) = 1.00. Since for most of the Strong
% Hierarchy case studies main effects were active with high probability.

% RWH for 200 models for Strong Hierarchy RWH Model

% 09/26/2005 by Jagmeet Singh

clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g');  % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d e');      % defining 2(5) Full Factorial Array for NF's



modelpara=6;    % Defining which model parameters we would be using for 2nd order model
% Basic WH(1);  Basic low w(2); Basic 2nd order(3); Fitted WH(4);   Fitted
% low order(5); Fitted 2nd order(6)

modelparameter=models(modelpara);      % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1);  s1=modelparameter(1,2); s2=modelparameter(1,3);
w1=modelparameter(1,4); w2=modelparameter(1,5);   p = 1.00;% defining parameters and
Changing 'p'

ncf=7;  % # of CF's
nnf=5;  % # of NF's
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;        % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

counter_p11 = 1;

for p11 = 0.01: 0.01: 1.00
X2(counter_p11) = p11;      % Store p11 values for final plot
p01 = p11;    % defining new probability parameters
p00 = p11;

  h1 = waitbar(0,'Running Models');
  for modelcounter=1:maxmodels      % To run a given number of models
    [bi,bij]=RWH_2ndorder(ncf,nnf,c,s1,w1,p,p11,p01,p00);   % Finding beta values for a given
model

    nfsetting1 = [-1*sign(bi(1:5));
      sign(bi(1:5))];  % defining compounded noise based on b-values as described above
    nfsetting2 = [-1;
      1];           % defining low and high setting of compounded noise




    % For 2(5) Full Factorial Noise Array
    clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
    index_resp_matrix = 1;  % Counter for response matrix
    for cfruns = 1:128
      for nfruns = 1:32
        x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
        x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
        sumij=0; sumijk=0;
        for i=1:(nnf+ncf)
          for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
          end
        end
        y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
        ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end


    varianc = var(y1')';
    STDev_1 = varianc.^0.5;      % Stdev for each CF setting
    std_base_1 = STDev_1(1,1);
    op_std_1 = min(STDev_1);   % Finding least Stdev
    PredMin_1 = sortrows([STDev_1 cfsetting],1);
    OCF_N1 = PredMin_1(1,2:8);
```

```
% For Compound Noise at 2 levels
clear ResponseMatrix_c;  % Response Matrix = [NF CF Y]
index_resp_matrix = 1;  % Counter for response matrix
for cfruns = 1:128
   for nfruns = 1:2
      x(1,1:5)=nfsetting1(nfruns,:); % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);   % Defining CF Setting
      sumij=0; sumijk=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end
      y_c(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
      ResponseMatrix_c(index_resp_matrix,:)=[nfsetting2(nfruns,:) cfsetting(cfruns,:)
y_c(cfruns,nfruns)];
      index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
   end
end


varianc = var(y_c')';
STDev_c = varianc.^0.5;    % Stdev for each CF setting
std_base_c = STDev_c(1,1);
op_std_c = min(STDev_c);   % Finding Least STDev_1
PredMin_c = sortrows([STDev_c cfsetting],1);
OCF_c = PredMin_c(1,2:8);

% Finding Optimal Standard Deviation from Compound Noise
for cfruns = 1:128
   if OCF_c == PredMin_1(cfruns, 2:8);
      Opt_c = PredMin_1(cfruns,1);
   end
end
```

```
    % Determining the Optimal Standard Deviation from Noise Strategy 1
    Opt_1 = op_std_1;

    std_base = mean(STDev_1);    % Base Stdev is taken as mean of all STDev's


    % Storing Improvement Ratios for Compound Noise
    std_fraction4(modelcounter) = ((std_base - Opt_c)/(std_base - Opt_1 + 1e-10));


    waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d for p11, p01, p00 =
%.2f',modelcounter, p11))
  end
  close(h1);  % Close waitbar

  improvement_ratio_mean(counter_p11) = mean(std_fraction4);      % Finding Improvement
Ratio for given p11
  improvement_ratio_median(counter_p11) = median(std_fraction4);      % Finding Improvement
Ratio for given p11
  counter_p11 = counter_p11 + 1;    % Increasing the Counter

end


% saving workspace
save variables;

clear; clc;
load variables;    % to remove previous data and upload the current data

% Plotting Improvement Ratio Mean vs P11, P01, P00

t2 = polyfit(X2, improvement_ratio_mean, 3);      % Fitting a 3rd order polynomial
y2 = polyval(t2,X2);
hold on;
plot(X2, improvement_ratio_mean, '.');
plot(X2, y2,'k','LineWidth',1, 'Marker', '+', 'MarkerEdgeColor','k',...
            'MarkerFaceColor','k',...
            'MarkerSize', 2);
xlabel('p_1_1, p_0_1, p_0_0', 'FontSize', 11);
ylabel('Mean Improvement Ratio',  'FontSize', 11);
title('Mean Improvement Ratio vs Density of Effects for RWH Model', 'FontSize',12);
ylim([0 1]);
hgsave('mean_improvement_ratio');
hold off;
figure;
```

```
% Plotting Improvement Ratio Median vs P11, P01, P00

t3 = polyfit(X2, improvement_ratio_median, 3);      % Fitting a 3rd order polynomial
y3 = polyval(t3,X2);
hold on;
plot(X2, improvement_ratio_median, '.');
plot(X2, y3,'k','LineWidth',1, 'Marker', '+', 'MarkerEdgeColor','k',...
            'MarkerFaceColor','k',...
            'MarkerSize', 2);
xlabel('p_1_1, p_0_1, p_0_0', 'FontSize', 11);
ylabel('Median Improvement Ratio',  'FontSize', 11);
title('Median Improvement Ratio vs Density of Effects for RWH Model', 'FontSize',12);
ylim([0 1]);
hgsave('median_improvement_ratio');
hold off;
```

6.1

```
% We first assume the model parameters we want to use in RWH Weak Hierarchy
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Compounding is done by first finding the signs of b's for noise variables
% and low level of noise is set as the ones having signs opposite to that
% of their b-values and vice-versa for high setting. (INDEPENDENT NOISES).

% We will find optimal control factor setting for compounded noise using
% Transmitted Variance Model. And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and RWH
% model determines how active they are.

% Find Improvement Ratio for each value of p11, p01, p00, p111, p011
% p001 and p000 from 0.01 to 1.00
% The p (prob. of active main effects) = 0.95. Since for most of the Strong
% Hierarchy case studies main effects were active with high probability.

% BASIC WH for 200 models for Weak Hierarchy RWH Model

% 10/16/2005 by Jagmeet Singh

clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g');  % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d e');      % defining 2(5) Full Factorial Array for NF's



modelpara=1;   % Defining which model parameters we would be using for Weak Hierarchy
model
% Basic WH(1);  Basic low w(2); Basic 2nd order(3); Fitted WH(4);   Fitted
% low order(5); Fitted 2nd order(6)
modelparameter=models(modelpara);      % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1);     s1=modelparameter(1,2);     s2=modelparameter(1,3);
w1=modelparameter(1,4);    w2=modelparameter(1,5);     p=0.95;% defining parameters and
Changing 'p'
% defining parameters

ncf=7;  % # of CF's
nnf=5;  % # of NF's
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;        % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

counter_p11 = 1;

for p11 = 0.01: 0.01: 1.00
  X2(counter_p11) = p11;      % Store p11 values for final plot
  p01 = p11;     % defining new probability parameters
  p00 = p11;     p111 = p11;    p011 = p11;    p001 = p11;    p000 = p11;


  h1 = waitbar(0,'Running Models');
  for modelcounter=1:maxmodels      % To run a given number of models
    [bi,bij,bijk]=RWH_3rdorder(ncf,nnf,c,s1,s2,w1,p,p11,p01,p00,p111,p011,p001,p000);   %
Finding beta values for a given model

    nfsetting1 = [-1*sign(bi(1:5));
      sign(bi(1:5))];  % defining compounded noise based on b-values as described above
    nfsetting2 = [-1;
      1];           % defining low and high setting of compounded noise




    % For 2(5) Noise Array
    clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
    index_resp_matrix = 1;  % Counter for response matrix
    for cfruns = 1:128
      for nfruns = 1:32
        x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
        x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
        sumij=0; sumijk=0;
        for i=1:(nnf+ncf)
          for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
          end
        end
        for i=1:(nnf+ncf)
          for j=i+1:(nnf+ncf)
            for k=j+1:(nnf+ncf)
              sumijk=sumijk+bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
            end
          end
        end
        y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
        ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
```

```
    end
  end


  varianc = var(y1')';
  STDev_1 = varianc.^0.5;      % Stdev for each CF setting
  std_base_1 = STDev_1(1,1);
  op_std_1 = min(STDev_1);    % Finding least Stdev
  PredMin_1 = sortrows([STDev_1 cfsetting],1);
  OCF_N1 = PredMin_1(1,2:8);




  % For Compound Noise at 2 levels
  clear ResponseMatrix_c;  % Response Matrix = [NF CF Y]
  index_resp_matrix = 1;  % Counter for response matrix
  for cfruns = 1:128
    for nfruns = 1:2
      x(1,1:5)=nfsetting1(nfruns,:); % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
      sumij=0; sumijk=0;
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
        end
      end
      for i=1:(nnf+ncf)
        for j=i+1:(nnf+ncf)
          for k=j+1:(nnf+ncf)
            sumijk=sumijk+bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
          end
        end
      end
      y_c(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
      ResponseMatrix_c(index_resp_matrix,:)=[nfsetting2(nfruns,:) cfsetting(cfruns,:)
y_c(cfruns,nfruns)];
      index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
    end
  end


  varianc = var(y_c')';
  STDev_c = varianc.^0.5;     % Stdev for each CF setting
  std_base_c = STDev_c(1,1);
  op_std_c = min(STDev_c);    % Finding Least STDev_1
  PredMin_c = sortrows([STDev_c cfsetting],1);
  OCF_c = PredMin_c(1,2:8);


  % Finding Optimal Standard Deviation from Compound Noise
```

```
    for cfruns = 1:128
      if OCF_c == PredMin_1(cfruns, 2:8);
        Opt_c = PredMin_1(cfruns,1);
      end
    end




    % Determining the Optimal Standard Deviation from Noise Strategy 1
    Opt_1 = op_std_1;

    std_base = mean(STDev_1);    % Base Stdev is taken as mean of all STDev's


    % Storing Improvement Ratios for Compound Noise
    std_fraction4(modelcounter) = ((std_base - Opt_c)/(std_base - Opt_1 + 1e-10));


    waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d for all p =
%.2f',modelcounter, p11))
  end
  close(h1);  % Close waitbar

  improvement_ratio_mean(counter_p11) = mean(std_fraction4);      % Finding Improvement
Ratio for given p11
  improvement_ratio_median(counter_p11) = median(std_fraction4);      % Finding Improvement
Ratio for given p11
  counter_p11 = counter_p11 + 1;     % Increasing the Counter

end


% saving workspace
save variables;


clear; clc;
load variables;    % to remove previous data and upload the current data

% Plotting Improvement Ratio Mean vs P11, P01, P00, P111, P011, P001, P000

t2 = polyfit(X2, improvement_ratio_mean, 3);       % Fitting a 3rd order polynomial
y2 = polyval(t2,X2);
hold on;
plot(X2, improvement_ratio_mean, '.');
plot(X2, y2,'LineWidth',1, 'Marker', '+', 'MarkerEdgeColor','k',...
  'MarkerFaceColor','k',...
  'MarkerSize', 2);
```

```
xlabel('p_1_1, p_0_1, p_0_0, p_1_1_1, p_0_1_1, p_0_0_1, p_0_0_0 (Effect Density)', 'FontSize',
11);
ylabel('Mean Improvement Ratio', 'FontSize', 11);
title('Mean Improvement Ratio vs Density of Effects for RWH Model', 'FontSize',12);
ylim([0 1]);
hgsave('mean_improvement_ratio');
hold off;
figure;


% Plotting Improvement Ratio Median vs P11, P01, P00

t3 = polyfit(X2, improvement_ratio_median, 3);     % Fitting a 3rd order polynomial
y3 = polyval(t3,X2);
hold on;
plot(X2, improvement_ratio_median, '.');
plot(X2, y3,'LineWidth',1, 'Marker', '+', 'MarkerEdgeColor','k',...
   'MarkerFaceColor','k',...
   'MarkerSize', 2);
xlabel('p_1_1, p_0_1, p_0_0, p_1_1_1, p_0_1_1, p_0_0_1, p_0_0_0 (Effect Density)', 'FontSize',
11);
ylabel('Median Improvement Ratio', 'FontSize', 11);
title('Median Improvement Ratio vs Density of Effects for RWH Model', 'FontSize',12);
ylim([0 1]);
hgsave('median_improvement_ratio');
hold off;
```

7.1

```
function [M,V,VOUT]=opamp_IA
% This matlab code is used to Vout for the OpAmp for all the control factor
% settings given in Phadke table 8.4 and table 8.5. Here each setting of
% control factor is being fed to opamp_noiseOA.m and the calculation
% proceeds from there. -Rest of the Code is provided by Goeff Reber

% There is no input for this. And the output for the code is Vout for each
% noise factor setting for a given control factor setting and the results
% are stored in file.

% OUTPUT: Vout(CFsetting, NFsetting)
% [Mean,Variance,Voffset-output]=opamp_IA

% Changed for Verification of RWH Model
% 04/08/2004 by Jagmeet Singh

clear;
clc;

% Defining the Control Factor Setting for the differential Op-Amp Circuit
CFlevel=[  35.5e3   71e3;
        7.5e3    15e3;
        1.25e3   2.5e3;
        10e-6    20e-6;
        10e-6    20e-6;];

% Defining full factorial Inner Array
IA=ff2n(5)+1;

% Running experiments for each control factor settings

for i=1:32


   % Determine control factor settings for the current experiment
   for j=1:5
      CF(j)=CFlevel(j,IA(i,j));
   end

   % Defining the variable for control factor setting with is being passed
   % to opamp_noiseOA.m
   Xc=CF;

   [Vmean, Vvariance, Vout]=opamp_noiseOA(Xc,i);

   % Variables to store values for each CF setting
   MEANofV(i,1)=Vmean;
   VARIANCEofV(i,1)=Vvariance;
```

```
   SNRatio(i,1)=(-10)*(log10(Vvariance/1000000));
   VOUTforCF(i,:)=Vout';

end

% save('Vout_CF.xls','VOUTforCF','-tabs','-ascii');
% save('mean.xls','MEANofV','-ascii');
% save('variance.xls','VARIANCEofV','-ascii');

% Saving mean, variance and Vout variables
M = MEANofV;
V = VARIANCEofV;

% The rows of VOUT are for a given control factor setting
VOUT = VOUTforCF;

% To save the workspace so as to work on it later on
IA = IA*2 - 3;
OA=fracfact('a b c d e f abc abd acd bcd abe ace bce ade abf adf bdf aef cef def bcdef');
OA=(OA+3)./2;
OA = OA*2 - 3;

save WORKSPACE IA M V SNRatio VOUT OA
end
```

7.2

```
function varargout=opamp_noiseOA(Xc,innerarray)
%Uses an L36 orthogonal array to compute the variation in offseet voltage
%for the opamp problem at a given control factor setting.
%INPTUS:
% Xc = control factor settings to test at
%OUTPUTS:
% mean = average offset voltage for the given control factors settings
% variance (optional) = variance in offset voltage for the given control factors
% eta (optional) = S/N ratio (signed-target type) for the given Xc

% Changed for Verification of RWH Model
% 04/08/2004 by Jagmeet Singh

X=zeros(21,1); %The entries of X change for different experiments

% if isempty(Xc) %No control parameter settings supplied
 %  Xc(1,1)=71e3;  Xc(2,1)=15e3;    Xc(3,1)=2.5e3;
 %  Xc(4,1)=20e-6;  Xc(5,1)=20e-6;
% end

Xn=Xc;
%Non control parameters
Xn(11)=.9817;  Xn(13)=.971;   Xn(15)=.975;
Xn(16)=3e-13;  Xn(18)=6e-13;  Xn(20)=6e-13;
Xn(21)=298;
%The entries of Xn are static and needed to compute X

%Define Testing Levels: the 1st 10 factors have 2 levels
N=[ .9967   1.0033;
   .93     1.07;
   .93     1.07;
   .98     1.02;
   .98     1.02;
   .9933   1.0067;
   .9933   1.0067;
   .9933   1.0067;
   .9933   1.0067;
   .9933   1.0067;
   .99     1;
   .998    1;
   .99     1;
   .998    1;
   .99     1;
   .45     1;
   .92     1;
   .45     1;
   .92     1;
   .67     1;
   .94     1;   ];
```

```
%Create the 64 run 2(IV)21-15 Othogonal Array
OA=fracfact('a b c d e f abc abd acd bcd abe ace bce ade abf adf bdf aef cef def bcdef');
OA=(OA+3)./2;

   %Begin experiments
h=waitbar(0,'Running Experiments in Outer Array');
for i=1:size(OA,1)
   %Determine factor levels for current experiment
   for j=1:5
         X(j)=Xn(j)*N( j, OA(i,j) );
   end
   j=6;   X(j)=X(1)*N( j, OA(i,j) );  %
   j=7;   X(j)=X(1)*N( j, OA(i,j) )/3.55;  %
   j=8;   X(j)=X(1)*N( j, OA(i,j) )/3.55;  %
   j=9;   X(j)=X(2)*N( j, OA(i,j) );  %
   j=10;   X(j)=X(3)*N( j, OA(i,j) );  %
   j=11;   X(j)=Xn(j)*N( j, OA(i,j) );
   j=12;   X(j)=X(11)*N( j, OA(i,j) ); %
   j=13;   X(j)=Xn(j)*N( j, OA(i,j) );
   j=14;   X(j)=X(13)*N( j, OA(i,j) ); %
   j=15;   X(j)=Xn(j)*N( j, OA(i,j) );
   j=16;   X(j)=Xn(j)*N( j, OA(i,j) );
   j=17;   X(j)=X(16)*N( j, OA(i,j) ); %
   j=18;   X(j)=Xn(j)*N( j, OA(i,j) );
   j=19;   X(j)=X(18)*N( j, OA(i,j) ); %
   j=20;   X(j)=Xn(j)*N( j, OA(i,j) );
   j=21;   X(j)=Xn(j)*N( j, OA(i,j) );
   waitbar(i/(size(OA,1)), h, sprintf('Running Inner array experiment %d for outer array %d',
innerarray,i) )
   Voff(i,1)=opamp_red(X)*1e3; %Runs the experiment
end
close(h)    %Close the waitbar

%Compute the mean, variance, and eta
mean = mean(Voff);
variance = var(Voff);
if nargout == 3
   eta = 10*log10(variance) ;
end

varargout{1}=mean;
t(1)=mean;
varargout{2}=variance;
t(2)=variance;
varargout{3}=Voff;
%varargout{3}=eta;
```

7.3

```
function [vout,converged]=opamp_red(X)
%This function computes the offset voltage of an operational amplifier used
%in coin-operated telephones.  The amplifier is described in Phadke's
%"Quality Engineering Using Robust Design".  The governing equations for
%the circuit were determined by Dr. Frey.  The input X is a 21-element
%vector of inputs describing 20 circuit elements and the ambient
%temperature, which are used to solve a reduced non-linear set of
%equations to determine voltage offset.
%OUTPUT:
% vout = offset voltage in Volts
% converged = boolean indicating whether the governing equations where
%satisfactorially solved.

% From Goeff Reber

%Constants
K=1.380658e-23;    %Boltzmann's Constant
q=1.602177e-19;    %Electron charge

%Unpack the elements of X
Rrfm=X(1);    Rrpem=X(2);    Rrnem=X(3);    Icpcs=X(4);    Iocs=X(5);
Rrfp=X(6);    Rrim=X(7);    Rrip=X(8);    Rrpep=X(9);    Rrnep=X(10);
Afpm=X(11);    Afpp=X(12);    Afnm=X(13);    Afnp=X(14);    Afno=X(15);
Siepm=X(16);    Siepp=X(17);    Sienm=X(18);    Sienp=X(19);    Sieno=X(20);
T=X(21);

%Determine the voltage values V which solve the reduced set of equations
v(1)=.4190;  v(2)=.4370;    v(3)=.4380;
% v=.4*ones(3,1);
%V = newtsearch( @opamp_goveq_red, 'true', v, 1e-12, 1500, [], X ); %home-made minimizer
opts = optimset('Display','off','TolX',1e-12,'MaxIter',1500,'TolFun',1e-12) ;
exitflag = 0 ;
maxiter = optimget(opts,'MaxIter');
while exitflag==0 & maxiter<1e5    %Make sure optimizer converges to an answer
   home
   [v,Fval,exitflag] = fsolve(@opamp_goveq_red,v,opts,X);
   maxiter = optimget(opts,'MaxIter');
   opts = optimset(opts,'MaxIter',maxiter*1.25);
end
if exitflag==0
   converged=false;
else
   converged='true';
end
V=v;   %Use the current guess to find Voffset

%Unpack elements of V
Vbe4=V(1);    Vbe1=V(2);    Vbe3=V(3);
```

```
%Find the offset voltage, vout (long equation broken into several terms)
term0=Icpcs - Siepm*Afpm + 2*Siepp - Siepp*Afpp + 2*Siepm + Siepm*Afpm*exp(Vbe1*q/K/T) -
2*Siepm*exp(Vbe1*q/K/T);
term0=Siepp*( term0/( 2*Siepp - Siepp*Afpp ) - 1 );
term1=Rrfp*( (term0-Sienp*( exp(Vbe4*q/K/T) - 1 ))/(1-Afno) - Iocs - (1-Afpm)*Siepm*(
exp(Vbe1*q/K/T) - 1 ) - (1-Afpp)*term0 );
term2=Rrip*( (term0-Sienp*( exp(Vbe4*q/K/T) - 1 ))/(1-Afno) - Iocs - (1-Afpm)*Siepm*(
exp(Vbe1*q/K/T) - 1 ) );
term3=Rrim*term2/Rrip;
term4=Rrfm*( (term0-Sienp*( exp(Vbe4*q/K/T) - 1 ))/(1-Afno) - Iocs );
vout= term1 + term2 + term3 + term4 ;
vout=-vout;
```

7.4

```
function F=opamp_goveq_red(V, X)  %Reduced Set
%This function provides the governing equations for the differential opamp
%problem described in function "opamp".  The equations are non-linear.
%Input vector V (length=3) contains three circuit voltages, X is 21 element
%vector describing circuit paramaters.  This function is meant to be called
%by a minimization routine which will select the values of V to minimize
%the sum of squares of three equations (in which X are parameters and V are
%variables whose values need to be determined)

% From Goeff Reber

%Constants
K=1.380658e-23;    %Boltzmann's Constant
q=1.602177e-19;    %Electron charge

%Unpack the elements of V
Vbe4=V(1);     Vbe1=V(2);     Vbe3=V(3);

%Unpack the elements of X
Rrfm=X(1);     Rrpem=X(2);    Rrnem=X(3);    Icpcs=X(4);    Iocs=X(5);
Rrfp=X(6);     Rrim=X(7);     Rrip=X(8);     Rrpep=X(9);    Rrnep=X(10);
Afpm=X(11);    Afpp=X(12);    Afnm=X(13);    Afnp=X(14);    Afno=X(15);
Siepm=X(16);   Siepp=X(17);   Sienm=X(18);   Sienp=X(19);   Sieno=X(20);
T=X(21);

F(1,1)= (1-Afnm)*Sienm*( exp(Vbe3*q/K/T) -1 ) - Siepm*( exp(Vbe1*q/K/T) - 1 ) ...
   + Sienm*( exp(Vbe3*q/K/T) -1 ) + (1-Afnp)*Sienp*( exp(Vbe4*q/K/T) - 1 );
F(2,1)= Rrnem*( Sienm*( exp(Vbe3*q/K/T) - 1 ) + (1-Afnm)*Sienm*( exp(Vbe3*q/K/T) - 1 ) ) ...
  -Rrnep*( Sienp*( exp(Vbe4*q/K/T) - 1 ) + (1-Afnp)*Sienp*( exp(Vbe4*q/K/T) - 1 ) ) ...
  - Vbe4 + Vbe3;
%The third equation is really large, so it's broken up into several terms
term0=Icpcs - Siepm*Afpm + 2*Siepp - Siepp*Afpp + 2*Siepm + Siepm*Afpm*exp(Vbe1*q/K/T) -
2*Siepm*exp(Vbe1*q/K/T);
term0=Siepp*( term0/( 2*Siepp - Siepp*Afpp ) - 1 );
term1=Rrip*( (term0-Sienp*( exp(Vbe4*q/K/T) - 1 ))/(1-Afno) - Iocs - (1-Afpm)*Siepm*(
exp(Vbe1*q/K/T) - 1 ) );
term2=Rrim*(term1/Rrip);
term3=-Vbe1 - Rrpem*( Siepm*( exp(Vbe1*q/K/T) - 1 ) + (1-Afpm)*Siepm*( exp(Vbe1*q/K/T) - 1 )
);
term4=Rrpep*( term0 + (1-Afpp)*term0 );
term5=Icpcs - Siepm*Afpm + 2*Siepp - Siepp*Afpp + 2*Siepm + Siepm*Afpm*exp(Vbe1*q/K/T) -
2*Siepm*exp(Vbe1*q/K/T);
term5=K*T*log( term5/(2*Siepp-Siepp*Afpp) )/q;
% if ~isreal(term5)
%    disp('Imaginary value found, press any key to continue')
%    pause
% end
F(3,1)= term1 + term2 + term3 + term4 + abs(term5);
```

7.5

```
function [sumsq, meansumsq]=sumsquare(obs)
% This program finds the Sum of Square and Mean Sum of Square for Control
% Factor Setting for Opamp.

% INPUT is a matrix in which the observations are sorted according to the
% treatments. Each treatment constitutes a column.

% OUTPUT contains the Sum of Squares and Mean Sum of Squares for the
% treatments

% 02/07/2004 - Jagmeet Singh

[n a]=size(obs);    % Find the size of the observation matrix
ytreatments = sum(obs); % Find the sum of observation for each treatment
ytotal = sum(ytreatments);  % Find the sum of all obervations

sum1=0;
for i=1:a
   sum1=sum1+ (ytreatments(i)^2);
end

sumsq = (1/n)*sum1 - (1/(n*a))*(ytotal^2);
meansumsq = sumsq/(a-1);
```

7.6

```
function [totsumsq, totmeansumsq]=totalsum(obs)
% This program finds the  Total Sum of Square and Total Mean Sum of Square for Control
% Factor Setting for Opamp.

% INPUT is a matrix in which the observations all the observations are given (eg: SNRatio and
Mean)

% OUTPUT contains the  Total Sum of Squares and Total Mean Sum of Squares for the
% treatments

% 02/07/2004 - Jagmeet Singh

[m]=size(obs);    % Find the size of the observation matrix
ytotal = sum(obs);  % Find the sum of all obervations

sum1=0;
for i=1:m
   sum1=sum1+ (obs(i)^2);
end

totsumsq = sum1 - (1/m(1,1))*(ytotal^2);
totmeansumsq = totsumsq/(m(1,1)-1);
```

7.7

```
% This program takes input from the opamp_IA.m and plots the graphs and the
% tables to find the optimal setting of the Control Factors for the Opamp
% for a given Noise Factor Strategy
% 02/07/2004 - Jagmeet Singh

clear
clc

load WORKSPACE;

% To get the setting for the control factors given by IA
rfm=IA(:,1);
rpem=IA(:,2);
rnem=IA(:,3);
cpcs=IA(:,4);
ocs=IA(:,5);

%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% Doing analysis for SN Ratio
% Matrix avgSN contains the averages for each setting of a given Control
% Factor

% To store the sorted result for RFM
sortrfm = sortrows([rfm SNRatio],1);
dummy=[sortrfm(1:12,2) sortrfm(13:24,2) sortrfm(25:36,2)];  % Stores values for each CF setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgSN(1,:)=dummymean;  % Stores values for RFM CF
[SumSq(1,:),MeanSq(1,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To store the sorted result for RPEM
sortrpem = sortrows([rpem SNRatio],1);
dummy=[sortrpem(1:12,2) sortrpem(13:24,2) sortrpem(25:36,2)];  % Stores values for each CF
setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgSN(2,:)=dummymean;  % Stores values for RPEM CF
[SumSq(2,:),MeanSq(2,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To store the sorted result for RNEM
sortrnem = sortrows([rnem SNRatio],1);
dummy=[sortrnem(1:12,2) sortrnem(13:24,2) sortrnem(25:36,2)];  % Stores values for each CF
setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgSN(3,:)=dummymean;  % Stores values for RNEM CF
[SumSq(3,:),MeanSq(3,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square
```

```
% To store the sorted result for CPCS
sortcpcs = sortrows([cpcs SNRatio],1);
dummy=[sortcpcs(1:12,2) sortcpcs(13:24,2) sortcpcs(25:36,2)];  % Stores values for each CF
setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgSN(4,:)=dummymean;  % Stores values for CPCS CF
[SumSq(4,:),MeanSq(4,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To store the sorted result for OCS
sortocs = sortrows([ocs SNRatio],1);
dummy=[sortocs(1:12,2) sortocs(13:24,2) sortocs(25:36,2)];  % Stores values for each CF setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgSN(5,:)=dummymean;  % Stores values for OCS CF
[SumSq(5,:),MeanSq(5,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To find the Total Sum of Square and Total Mean Square for SNRatio
[SNtotsum,SNtotmsq] = totalsum(SNRatio);

% To find Errors DOF, Sum of Square and Mean Square
SSe = SNtotsum - sum(SumSq);   % Error Sum of Square
[n a] = size(dummy);
dof_error = n*a - 5*(a-1) - 1;
dof_total = n*a - 1;
Msse = SSe/dof_error;  % Defining Mean Square for Error

Fratio = MeanSq/Msse;  % Defining F Ratio for the CF's
Overall_mean = mean(SNRatio);

% Presenting the results in Tabular form
Table = zeros(7,7); % Formatting OUTPUT Table

for j=1:3
   Table(:,j) = [ avgSN(:,j)' Inf Inf]';
end

Table(:,4) = [a-1 a-1 a-1 a-1 a-1 dof_error dof_total]';
Table(:,5) = [SumSq(:,1)' SSe SNtotsum]';
Table(:,6) = [MeanSq(:,1)' Msse SNtotmsq]';
Table(:,7) = [Fratio(:,1)' Inf Inf]';

colheads = ['Factor        '
        'Avg SN-Level 1  '
        'Avg SN-Level 2  '
        'Avg SN-Level 3  '
        '   DOF       '
        'Sum Of Square   '
        ' Mean Square    '
        '   F         '];
```

```
rowheads = ['A. RFM              ';
        'B. RPEM             ';
        'C. RNEM             ';
        'D. CPCS             ';
        'E. OCS              ';
        'Error           ';
        'Total           '];


% Create cell array version of table
atab = num2cell(Table);
for i=1:size(atab,1)
  for j=1:size(atab,2)
    if (isinf(atab{i,j}))
      atab{i,j} = [];
    end
  end
end
atab = [cellstr(strjust(rowheads, 'left')), atab];
atab = [cellstr(strjust(colheads, 'left'))'; atab];

  wtitle = 'ANALYSIS of S/N Ratio';
  ttitle = 'TABLE';

  digits = [-1 -1 -1 -1 0 1 1 0];
  statdisptable(atab, wtitle, ttitle, '', digits);

% To plot S/N Ratio graph
h = figure;
TITLE('S/N Ratio Graph');
XLABEL('Factors');
YLABEL('S/N Ratio, in dB');

h2 = text(2,min(min(avgSN)),'RFM','FontWeight','bold')
text(5,min(min(avgSN)),'RPEM','FontWeight','bold');
text(8,min(min(avgSN)),'RNEM','FontWeight','bold');
text(11,min(min(avgSN)),'CPCS','FontWeight','bold');
text(14,min(min(avgSN)),'OCS','FontWeight','bold');

AXIS([0 16 min(min(avgSN))-2 max(max(avgSN))+2]);
X = [0;16]; Y = [Overall_mean;Overall_mean];
h1 = line(X,Y,'Color','k');
get(h1)

for i=1:5
   X = [3*i-2;3*i-1;3*i];Y= (avgSN(i,:))';
   line(X,Y,'LineWidth',2,'Marker','*')
end
```

```
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------
% Doing analysis for Mean
% Matrix avgM contains the averages for each setting of a given Control
% Factor

% To store the sorted result for RFM
sortrfm = sortrows([rfm M],1);
dummy=[sortrfm(1:12,2) sortrfm(13:24,2) sortrfm(25:36,2)];  % Stores values for each CF setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgM(1,:)=dummymean;  % Stores values for RFM CF
[SumSq(1,:),MeanSq(1,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To store the sorted result for RPEM
sortrpem = sortrows([rpem M],1);
dummy=[sortrpem(1:12,2) sortrpem(13:24,2) sortrpem(25:36,2)];  % Stores values for each CF
setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgM(2,:)=dummymean;  % Stores values for RPEM CF
[SumSq(2,:),MeanSq(2,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To store the sorted result for RNEM
sortrnem = sortrows([rnem M],1);
dummy=[sortrnem(1:12,2) sortrnem(13:24,2) sortrnem(25:36,2)];  % Stores values for each CF
setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgM(3,:)=dummymean;  % Stores values for RNEM CF
[SumSq(3,:),MeanSq(3,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To store the sorted result for CPCS
sortcpcs = sortrows([cpcs M],1);
dummy=[sortcpcs(1:12,2) sortcpcs(13:24,2) sortcpcs(25:36,2)];  % Stores values for each CF
setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgM(4,:)=dummymean;  % Stores values for CPCS CF
[SumSq(4,:),MeanSq(4,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To store the sorted result for OCS
sortocs = sortrows([ocs M],1);
dummy=[sortocs(1:12,2) sortocs(13:24,2) sortocs(25:36,2)];  % Stores values for each CF setting
dummymean=mean(dummy);  % Stores the mean for each CF setting
avgM(5,:)=dummymean;  % Stores values for OCS CF
[SumSq(5,:),MeanSq(5,:)]=sumsquare(dummy);  % Stores values for Sum of Sqaure and Mean
Sum of Square

% To find the Total Sum of Square and Total Mean Square for M
[SNtotsum,SNtotmsq] = totalsum(M);
```

```
% To find Errors DOF, Sum of Square and Mean Square
SSe = SNtotsum - sum(SumSq);    % Error Sum of Square
[n a] = size(dummy);
dof_error = n*a - 5*(a-1) - 1;
dof_total = n*a - 1;
Msse = SSe/dof_error;   % Defining Mean Square for Error

Fratio = MeanSq/Msse;   % Defining F Ratio for the CF's
Overall_mean = mean(M);

% Presenting the results in Tabular form
Table = zeros(7,7); % Formatting OUTPUT Table

for j=1:3
   Table(:,j) = [ avgM(:,j)' Inf Inf]';
end

Table(:,4) = [a-1 a-1 a-1 a-1 a-1 dof_error dof_total]';
Table(:,5) = [SumSq(:,1)' SSe SNtotsum]';
Table(:,6) = [MeanSq(:,1)' Msse SNtotmsq]';
Table(:,7) = [Fratio(:,1)' Inf Inf]';

colheads = ['Factor          '
         'Avg SN-Level 1  '
         'Avg SN-Level 2  '
         'Avg SN-Level 3  '
         '    DOF         '
         'Sum Of Square   '
         ' Mean Square    '
         '    F           '];

rowheads = ['A. RFM                ';
         'B. RPEM               ';
         'C. RNEM               ';
         'D. CPCS              ';
         'E. OCS              ';
         'Error              ';
         'Total              '];


% Create cell array version of table
atab = num2cell(Table);
for i=1:size(atab,1)
  for j=1:size(atab,2)
    if (isinf(atab{i,j}))
      atab{i,j} = [];
    end
  end
end
atab = [cellstr(strjust(rowheads, 'left')), atab];
atab = [cellstr(strjust(colheads, 'left'))'; atab];
```

```
  wtitle = 'ANALYSIS of Mean Offset Voltage';
  ttitle = 'TABLE';

  digits = [-1 -1 -1 -1 0 1 1 0];
  statdisptable(atab, wtitle, ttitle, '', digits);

% To plot S/N Ratio graph
h = figure;
TITLE('Mean Offset Voltage Graph');
XLABEL('Factors');
YLABEL('Mean Offset Voltage (10e-3 V)');
text(2,min(min(avgM)),'RFM','FontWeight','bold');
text(5,min(min(avgM)),'RPEM','FontWeight','bold');
text(8,min(min(avgM)),'RNEM','FontWeight','bold');
text(11,min(min(avgM)),'CPCS','FontWeight','bold');
text(14,min(min(avgM)),'OCS','FontWeight','bold');

AXIS([0 16 min(min(avgM))-2 max(max(avgM))+2]);
X = [0;16]; Y = [Overall_mean;Overall_mean];
h1 = line(X,Y,'Color','k');
get(h1)

for i=1:5
   X = [3*i-2;3*i-1;3*i];Y= (avgM(i,:))';
   line(X,Y,'LineWidth',2,'Marker','*')
end
```

7.8

```
% To find significant 3 way interactions we need to do regression on the
% results from the run. We will analyze the beta's using both Normal Plots
% and Lenth Method.

% Most of the variables have 3 levels, so we will analyze them in 2 steps.
% We will break 3 levels into 2 - 2 levels and find the betas to analyze.

% INPUT: Variables from the Runs of Madhav's Strategy with Different IA and
% OA

% OUTPUT: beta values and plotting Normal Plot and finding significant
% factors

% 04/08/2004 by Jagmeet Singh

clear;clc;

load WORKSPACE

t = 1;  % Index for rows of Regression Variable

for i=1:32
   for j=1:64

      % Reg_Var = [CF : NF : VOUT]
      % Reg_Var = [IA : OA : VOUT]
      Reg_Var(t,1:5) = IA(i,:);
      Reg_Var(t,6:26) = OA(j,:);
      Reg_Var(t,27) = VOUT(i,j);
      t = t + 1;

   end
end


%=========================
% ANALYSIS
%=========================

% To find beta values for main effects, 2 and 3 way interactions
Y(:,1) = Reg_Var(:,27)/1e305;
X(:,1) = ones(32*64,1);
dum_x(:,1:26) = Reg_Var(:,1:26);

% Finding 2way interactions terms
t=1;    % Index for storing the results in dum_x
   for i=1:5
```

```
    for j=i+1:5
        CxC(:,t)=dum_x(:,i).*dum_x(:,j);
        t=t+1;
    end
end
t=1;
for i=1:5
    for j=11:26
        CxN(:,t)=dum_x(:,i).*dum_x(:,j);
        t=t+1;
    end
end
t=1;
for i=11:26
    for j=i+1:26
        NxN(:,t)=dum_x(:,i).*dum_x(:,j);
        t=t+1;
    end
end

t=1;
% Finding 3way interaction terms
for i = 1:5
    for j = i+1:5
        for k = j+1:5
            CxCxC(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
            t = t+1;
        end
    end
end
t=1;
for i = 1:5
    for j = i+1:5
        for k = 11:26
            CxCxN(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
            t = t+1;
        end
    end
end
t=1;
for i = 1:5
    for j = 11:26
        for k = j+1:26
            CxNxN(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
            t = t+1;
        end
    end
end
t=1;
for i = 11:26
    for j = i+1:26
        for k = j+1:26
            NxNxN(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
```

```
            t = t+1;
        end
    end
end


way2 = [CxC CxN NxN];

way3 = [CxCxC CxCxN CxNxN NxNxN];

X = [X dum_x way2 way3];        % Defining regression coefficients
% Main effects col:  2-27
% 2 way effects col:  28-237   (CxC: 28-37)(CxN: 38-117)(NxN: 118-237)
% 3 way effects col:  238-1007  (CxCxC: 238-247)(CxCxN: 248-407)(CxNxN:
% 408-1007)(NxNxN: 1008-1567)

b1 = regress(Y,X);      % Finding the beta's

save WORKSPACE_b1 IA OA Reg_Var VOUT X Y dum_x way2 way3 b1;    % saving variables
for further use

% Implementing Lenth Method
bi = abs(b1);
s0 = 1.5*median(bi);

limit=2.5*s0;
t=1;
for i=1:1567
   if bi(i) <= limit
      dumm(t)=bi(i);
      t=t+1;
   end
end
PSE = 1.5*median(dumm);
m=1567;
ME = tinv(0.975,m/3)*PSE;
gamma=(1 + (0.95^(1/m)))/2.0;
SME = tinv(gamma,m/3)*PSE;
hold on;
plot(b1);
xlabel(' Contrasts ');
ylabel(' Estimated Contrasts');
plot([0 1200],[ME ME],'--r');
plot([0 1200],[-ME -ME],'--r');
plot([0 1200],[SME SME],'-.k');
plot([0 1200],[-SME -SME],'- .k');
plot([27 27],[max(b1) min(b1)], 'k','LineWidth',2);
plot([238 238],[max(b1) min(b1)], 'k','LineWidth',2);
title('Lenth-Bayes Plot for Verification-1');
text(27,max(b1)/1.2,'Two Way Effects');
text(400,max(b1)/1.2,'Three Way Effects');
```

```
text(1100,SME,'SME');
text(1100,-SME,'- SME');
text(1100,ME,'ME');
text(1100,-ME,'- ME');
```

8.1

```
function varargout=cstr(X,varargin)
%Model's a continuously stired tank reactor designed producing "reactant b"
%INPUTS:
% X = a vector of control parameters for the reaction process
% varargin (optional) = a scalar representing the mean of Rb for the given
%control parameters
%OUTPUTS:
% rb (optional) = the molecular rate of production of reactant b.  If
%varargin is not empty, then rb is changed to squared error from the mean.
% Ca (optional) = final concentration of reactant A
% Cb (optional) = final concentration of reactant B
% Ra (optional) = steady state temperature of the reacting tank
% F (optional) = volumetric flow rate into reacting tank

% From Goeff Reber

maxiter=1e4;
iter=1;

%Control Parameters:
Cai=X(1);   %Initial concentration of reactant A
Cbi=X(2);   %Initial concentration of reactant B
Ti=X(3);    %Temperature of the mixture entering the reacting tank
Q=X(4);     %Heat added to the reacting tank
V=X(5);     %Steady state volume liquid in the reacting tank
F=X(6);     %Volumetric flow rate into the reacting tank

%System Parameters:
k0a=8.4*10^5;   %min^-1
k0b=7.6*10^4;   %min^-1
Hra=-2.12*10^4; %J/mol
Hrb=-6.36*10^4; %J/mol
Ea=3.64*10^4;   %J/mol
Eb=3.46*10^4;   %J/mol
Cp=3.2*10^3;    %J/kg/K
R=8.314;    %J/mol/K
rho=1180;   %kg/m^3

T=Ti;   %Initial gues
t=V/F;  %Residence Time
pdT=1;  %allows entry into the loop
iter=0;
while pdT>1e-3
   Ca=Cai/( 1 + t*k0a*exp(-Ea/R/T) );
   Ra=-Ca*k0a*exp(-Ea/R/T);
   Cb=( Cbi - t*Ra )/( 1 + t*k0b*exp(-Eb/R/T) );
   Rb=-Ra - Cb*k0b*exp(-Eb/R/T);
   Tnew=( Q - V*(Ra*Hra+Rb*Hrb) )/F/rho/Cp + Ti;
   pdT=100*abs(Tnew-T)/T;
```

```
   T=T+.5*(Tnew-T);
   iter=iter+1;
   if iter>maxiter
%       display('Max number of iterations reached in computing T')
      break, break, return
   end
end

if isempty(varargin)
   rb=Rb*V;
else
   rb=( Rb*V - varargin{1} )^2;   %Variance is requested, input varargin{1} is the mean
end
varargout{1}=rb;
varargout{2}=Ca;
varargout{3}=Cb;
varargout{4}=Ra;
varargout{5}=Rb;
varargout{6}=T;
varargout{7}=F;
```

8.2

```
% To find significant 3-way interactions in CSTR model. We will run full
% factorial array on both Control Factors and Noise Factors. Control
% Factors base levels are chosen as one given in Diwekar Paper[1997].
% The upper level is chosen to be 20% more than the base level. Noise
% factors are chosen at 5% distance from the Control Factor Setting.

% The betas from the regression are analyzed by both Lenth Method and
% Normal Plots.

% Output: Lenth Plot

% 04/15/2004 by Jagmeet Singh

clear;clc;

Xbase = [3119.8 342.24 309.5 5.0e6 0.05 0.043]; % Base Level of CF's
Xup = 1.2*Xbase;    % Upper Levels are 20% more than Base Level
CF = [Xbase' Xup']; % Defining Control Factor array
IA = ff2n(6);       % Defining fullfactorial Inner Array
OA = ff2n(6);       % Full Factorial Outer Array

IA = IA+1;

% Finding Response for inner-outer array
h=waitbar(0);
for i=1:size(IA,1)
   level = IA(i,:);    % Level for CF to be chosen
   CF1 = CF(1, level(1));
   CF2 = CF(2, level(2));
   CF3 = CF(3, level(3));
   CF4 = CF(4, level(4));
   CF5 = CF(5, level(5));
   CF6 = CF(6, level(6));
   cf_level = [CF1 CF2 CF3 CF4 CF5 CF6];

   waitbar(i/size(IA,1), h, sprintf('Response for CF setting  %d',i));

   % Running OA for each Control Factor Setting
   for j=1:size(OA,1)
      oa_level = OA(j,:); % Level for NF to be chosen
      for dummy=1:6
         if oa_level(dummy)==0
            x_level(dummy) = cf_level(dummy)*0.95;  % -5% for low setting of NF
         else
            x_level(dummy) = cf_level(dummy)*1.05;  % 5% for higher setting of NF
         end
      end
      rb(i,j) = cstr(x_level); % Each row of rb corresponds to a CF setting
```

```
      % and columns to NF settings
   end
end
close(h);


IA = ff2n(6)*2 - 1;
OA = IA;              % IA and OA for regression
t = 1;  % Index for rows of Regression Variable

for i=1:size(IA,1)
   for j=1:size(OA,1)

      % Reg_Var = [CF : NF : rb]
      % Reg_Var = [IA : OA : rb]
      Reg_Var(t,1:6) = IA(i,:);
      Reg_Var(t,7:12) = OA(j,:);
      Reg_Var(t,13) = rb(i,j);
      t = t+1;
   end
end

%====================
% ANALYSIS
%====================

% To find beta values for main effects, 2 and 3 way interactions
Y(:,1) = Reg_Var(:,13);
X(:,1) = ones(size(IA,1)*size(OA,1),1);
dum_cf(:,1:6) = Reg_Var(:,1:6);
dum_nf(:,1:6) = Reg_Var(:,7:12);

% Finding 2 way interaction terms

t=1;
for i=1:6
   for j=i+1:6
      CxC(:,t)=dum_cf(:,i).*dum_cf(:,j);
      t=t+1;
   end
end

t=1;
for i=1:6
   for j=7:12
      CxN(:,t)=dum_cf(:,i).*dum_nf(:,j-6);
      t=t+1;
   end
end

% Finding 3 way interaction terms
```

```
t=1;
for i=1:6
   for j=i+1:6
      for k=j+1:6
         CxCxC(:,t) = dum_cf(:,i).*dum_cf(:,j).*dum_cf(:,k);
         t=t+1;
      end
   end
end

t=1;
for i=1:6
   for j=i+1:6
      for k=7:12
         CxCxN(:,t) = dum_cf(:,i).*dum_cf(:,j).*dum_nf(:,k-6);
         t=t+1;
      end
   end
end

t=1;
for i=1:6
   for j=7:12
      for k=j+1:12
         CxNxN(:,t) = dum_cf(:,i).*dum_nf(:,j-6).*dum_nf(:,k-6);
         t=t+1;
      end
   end
end

way2 = [CxC CxN];
way3 = [CxCxC CxCxN CxNxN];

X = [X dum_cf way2 way3];  % Defining Regression Coefficients
% Main CF effects col: 2-7
% 2 way effects col: 8-58 (CxC: 8-22)(CxN: 23-58)
% 3 way effects col: 59-258 (CxCxC: 59-78)(CxCxN: 79-168)(CxNxN: 169-258)

b = regress(Y,X);      % Finding the beta's

% Implementing Lenth Method
bi = abs(b);
s0 = 1.5*median(bi);

limit=2.5*s0;
t=1;
for i=1:258
   if bi(i) <= limit
      dumm(t) = bi(i);
      t=t+1;
```

```
   end
end
PSE=1.5*median(dumm);
m=258;
ME = tinv(0.975,m/3)*PSE;
gamma=(1 + (0.95^(1/m)))/2.0;
SME = tinv(gamma,m/3)*PSE;
hold on;
plot(b);
xlabel(' Contrasts ');
ylabel(' Estimated Contrasts');
plot([0 300],[ME ME],'--r');
plot([0 300],[-ME -ME],'--r');
plot([0 300],[SME SME],'-.k');
plot([0 300],[-SME -SME],'-.k');
plot([6 6],[max(b) min(b)], 'k','LineWidth',2);
plot([58 58],[max(b) min(b)], 'k','LineWidth',2);
title('Lenth-Bayes Plot for Verification');
text(7,max(b)/1.2,'Two Way Effects');
text(100,max(b)/1.2,'Three Way Effects');
text(300,SME,'SME');
text(300,-SME,'- SME');
text(300,ME,'ME');
text(300,-ME,'- ME');
figure;
normplot(b);
```

9.1

```
% Passive Neuron Model from "Non-Linear Parameter Estimation by Linear
% Association: Application to a Five-Parameter Passive Neuron Model", IEEE
% Transactions on Biomedical Engineering, Vol. 41, No.5, 1994.

% We will use the Somatic Shunt model of Neuron. We will look at its
% Laplace Transform Z(s) = V(0,s)/I(s). We will use the parameters given in
% the model and will determine the order of the system.

% Control Factors in this case are Ts and Tm. Noise factors are L, Rs, Rd,
% Rn.

% By Jagmeet Singh on 02/14/2005

clear;clc;
t = 3e-3;          % in secs. It is the time at which response is looked
s = 2*pi/t;        % Angular Frequency at which the response is looked




% Defining Control Factor Levels.
% Control Factors are Ts, Tm (in secs.)
CF_levels = [ 3e-3      5e-3;     % Ts
              5e-3      7e-3];    % Tm

% Defining Control Factor Array
CF_array = ff2n(2) + 1;
[m,n] = size(CF_array);
for i=1:m
   for j=1:n

      % Defining Inner Array for experiments
      Inner_array(i,j) = CF_levels(j,CF_array(i,j));
   end
end




% Defining Noise Array
% Noise Factors are L, Rs, Rd and Rn. Rs, Rd, Rn are in Ohm.
NF_levels = [0.7          1.1;        % L
             300e6        320e6;      % Rs
             120e6        160e6;      % Rd
              80e6         90e6;];    % Rn
```

```
% Defining Noise Array
NF_array = ff2n(4) + 1;
[m,n] = size(NF_array);
for i = 1:m
   for j = 1:n

      % Defining Outer Array Levels for the experiments
      Outer_array_levels(i,j) = NF_levels(j, NF_array(i,j));
   end
end




% Running Experiements at Inner and Outer Array Settings
[iarows, iacols] = size(Inner_array);
[oarows, oacols] = size(Outer_array_levels);
h1= waitbar(0);

for iaruns = 1:iarows
   waitbar(iaruns/iarows, h1, sprintf('Running Control Factor Experiment # %d',iaruns));

   % Setting the Control Factor Levels for Inner Array experiments
   Ts = Inner_array(iaruns,1);
   Tm = Inner_array(iaruns,2);

   index = 1;

   % Setting Outer Array Levels for Outer Array experiments
   for i = 1:oarows
      L = Outer_array_levels(i,1);
      Rs = Outer_array_levels(i,2);
      Rd = Outer_array_levels(i,3);
      Rn = Outer_array_levels(i,4);

      p = Rs/Rd;

      Z(iaruns, index) = (Rn * (p + 1))/(((Tm*s + 1)^(0.5)) * tanh(L * ((Tm*s + 1)^(0.5))) * p * coth(L)
+ (1 + Ts * s));

      index = index + 1;
   end
end

close(h1);
IA = CF_array*2 - 3;       % To define Inner array in -1,1 levels
OA = NF_array*2 - 3;       % To define Outer array in -1,1 levels

save WORKSPACE_1;
```

9.2

```
% To find significant 3-way interactions we need to do regression on the
% results from the runs. We will analyze beta's (the regression
% coefficients) using both Normal Plot and Lenth Method.

% INPUT: Variables from the runs of PNM Model with 2-levels factors in both
% CF and NF array.

% OUTPUT: beta values and plotting of Normal and Lenth Plot and finding
% significant factors.

% 2/14/2005 by Jagmeet Singh

clear;clc;
load WORKSPACE_1;
clear X;

t = 1;     % Index for rows of Regression Variable

for i = 1:4
   for j = 1:16
      % Reg_Var = [CF : NF : Z];
      % Reg_Var = [IA : OA : Z];
      Reg_Var(t,1:2) = IA(i,:);
      Reg_Var(t,3:6) = OA(j,:);
      Reg_Var(t,7) = Z(i,j)/1e7;
      t = t + 1;
   end
end




%%%%%%%%%%%%%%%%%%%
% ANALYSIS
%%%%%%%%%%%%%%%%%%%

% To find beta values for main effects, 2- and 3-way interactions
Y(:,1) = Reg_Var(:,7);
X(:,1) = ones(4*16, 1);
dum_x(:,1:6) = Reg_Var(:,1:6);

% Finding 2-way interaction terms
t=1;   % Index for storing results in dum_x
   for i=1:2
      for j=i+1:2
         CxC(:,t)=dum_x(:,i).*dum_x(:,j);
         t=t+1;
      end
```

```
    end
    t=1;
    for i=1:2
       for j=3:6
          CxN(:,t)=dum_x(:,i).*dum_x(:,j);
          t=t+1;
       end
    end
    t=1;
    for i=3:6
       for j=i+1:6
          NxN(:,t)=dum_x(:,i).*dum_x(:,j);
          t=t+1;
       end
    end

% Finding 3-way interactions
    t=1;
    for i=1:2
       for j=i+1:2
          for k=j+1:2
             CxCxC(:,t)=dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
             t=t+1;
             sprintf('Foul Value')
          end
       end
    end
    t=1;
    for i=1:2
       for j=i+1:2
          for k=3:6
             CxCxN(:,t)=dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
             t=t+1;
          end
       end
    end
    t=1;
    for i=1:2
       for j=3:6
          for k=j+1:6
             CxNxN(:,t)=dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
             t=t+1;
          end
       end
    end
    t=1;
    for i=3:6
       for j=i+1:6
          for k=j+1:6
             NxNxN(:,t)=dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
             t=t+1;
          end
       end
```

```
   end

way2 = [CxC CxN NxN];
way3 = [CxCxN CxNxN];

X = [ X dum_x way2 way3];      % defining Regression Matrix
% Main Effects col: 2:7

% 2-way Effects col: 8:22   (CxC: 8)   (CxN: 9:16)    (NxN: 17:22)

% 3-way Effects col: 23:38   (CxCxN: 23:26)      (CxNxN: 27:38)

b_part1 = regress(Y,X);    % Finding beta's

save WORKSPACE_part1;      % saving the variables for further use

normplot(b_part1);     % To plot normal plot for the regression coefficients
title('Normal Probability Plot -- Part 1');
hgsave('NormalPlot_part1')
figure;

% Implementing Lenth Method
bi = abs(b_part1);
s0 = 1.5*median(bi);

limit=2.5*s0;
t=1;
[m,n]=size(X);
for i=1:n
   if bi(i) <= limit
      dumm(t)=bi(i);
      t=t+1;
   end
end
PSE = 1.5*median(dumm);
ME = tinv(0.975,n/3)*PSE;
gamma=(1 + (0.95^(1/n)))/2.0;
SME = tinv(gamma,n/3)*PSE;
hold on;
bar(b_part1);
xlabel(' Contrasts ');
ylabel(' Estimated Contrasts');
plot([0 50],[ME ME],'--r');
plot([0 50],[-ME -ME],'--r');
plot([0 50],[SME SME],'-.k');
plot([0 50],[-SME -SME],'-.k');
plot([7 7],[max(b_part1) min(b_part1)], 'k','LineWidth',2);
plot([23 23],[max(b_part1) min(b_part1)], 'k','LineWidth',2);
title('Lenth-Bayes Plot for Verification -- Part 1');
text(6,max(b_part1)/1.2,'Two Way Effects');
text(25,max(b_part1)/1.2,'Three Way Effects');
```

```
text(45,SME,'SME');
text(45,-SME,'- SME');
text(45,ME,'ME');
text(45,-ME,'- ME');
hgsave('LenthPlot_part1');
```

10.1

```
% Hydrodynamic Journal Bearings -- Analytical Solutions is adopted from
% "Fundamentals of Fluid Film Lubrication", Second Edition, by Hamrock, B.,
% Schmid, S., and Jacobson, B. by Marcel Dekker.

% We will use Half Sommerfeld Solution for Infinitely Wide Journal Bearing.
% We will look at it Wr, which is resultant load per unit width in a
% Journal Bearing and is measured in N/m. We will use the parameters given
% in the solution for Wr and will determine the order of the system.

% Control Factors are: n0   rb
% Noise Factors are:   wb   c   e

% By Jagmeet Singh on 05/24/2005

clear;clc;




% Defining Control Factor Levels.
% Control Factors are n0 and rb
CF_levels = [ 0.002        0.005;        % n0
              40e-3        60e-3];       % rb


% Defining Control Factor Array
CF_array = ff2n(2) + 1;
[m , n] = size(CF_array);

for i = 1:m
   for j = 1:n

      % Defining Inner Array for experiments
      Inner_array(i,j) = CF_levels(j, CF_array(i,j));
   end
end




% Defining Noise Array
% Noise Factors are wb, c and e
NF_levels = [ 2*pi*100        2*pi*150;      % wb
              0.5e-3          0.8e-3;        % c
              0.2             0.7      ];    % e
```

```
% Defining Noise Array
NF_array = ff2n(3) + 1;
[m , n] = size(NF_array);

for i = 1:m
   for j = 1:n

      % Defining Outer Array Levels for the experiments
      Outer_array_levels(i,j) = NF_levels(j, NF_array(i,j));
   end
end




% Running Experiments at Inner and Outer Array Settings
[iarows, iacols] = size(Inner_array);
[oarows, oacols] =size(Outer_array_levels);
h1 = waitbar(0);


for iaruns = 1:iarows
   waitbar(iaruns/iarows, h1, sprintf('Running Control Factor Experiment # %d', iaruns));

   % Setting the Control Factor Levels for Inner Array Experiments
   n0 = Inner_array(iaruns, 1);
   rb = Inner_array(iaruns, 2);

   index = 1;

   % Setting Outer Array Levels for Outer Array Experiments
   for i = 1:oarows
      wb = Outer_array_levels(i,1);
      c = Outer_array_levels(i,2);
      e = Outer_array_levels(i,3);

      Wr(iaruns, index) = (n0*wb*rb*   ((rb/c)^2))*(   (6*e*  ((pi^2 - ((e^2)*(pi^2 - 4)))^(0.5)))/ ( (2 +
(e^2)) * ( 1 - (e^2))));

      index = index + 1;
   end
end

close(h1);
IA = CF_array*2 - 3;        % To define Inner Array in -1, 1 levels
OA = NF_array*2 - 3;        % To define Outer Array in -1, 1 levels

save WORKSPACE_1;
```

10.2

```
% To find significant 3-factor interactions we need to do regression on the
% results from the runs. We will analyze beta's (the regression
% coefficients) using both Normal Plot and Lenth Method.

% INPUT: Variables from the runs of Journal Bearing Half Sommerfeld
% Solution with 2-level factors in both CF and NF array.

% OUTPUT: beta values and plotting of Normal and Lenth Plot and finding
% significant factors.

% 05/25/2005 by Jagmeet Singh

clear;clc;
load WORKSPACE_1;
clear X;

t = 1;     % Index for rows of Regression Variable

for i = 1:4
   for j = 1:8
      % Reg_Var = [CF : NF : Wr];
      % Reg_Var = [IA : OA : Wr];
      Reg_Var(t,1:2) = IA(i,:);
      Reg_Var(t,3:5) = OA(j,:);
      Reg_Var(t,6) = Wr(i,j);
      t = t + 1;
   end
end




%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ANALYSIS
%%%%%%%%%%%%%%%%%%%%%%%%%%%

% To find beta values for main effects, 2- and 3- factor interactions
Y(:,1) = Reg_Var(:,6);
X(:,1) = ones(4*8,1);
dum_x(:,1:5) = Reg_Var(:,1:5);

% Finding 2-factor interaction terms
t = 1;     % Index for storing results in dum_x
for i = 1:2
   for j = i+1:2
      CxC(:,t) = dum_x(:,i).*dum_x(:,j);
      t = t + 1;
   end
```

```
end
t = 1;
for i = 1:2
   for j = 3:5
      CxN(:,t) = dum_x(:,i).*dum_x(:,j);
      t = t + 1;
   end
end
t = 1;
for i = 3:5
   for j = i+1:5
      NxN(:,t) = dum_x(:,i).*dum_x(:,j);
      t = t + 1;
   end
end


% Finding 3-factor interactions
t = 1;
for i = 1:2
   for j = i+1:2
      for k = j+1:2
         CxCxC(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
         t = t + 1;
         sprintf('Foul Value');
      end
   end
end
t = 1;
for i = 1:2
   for j = i+1:2
      for k = 3:5
         CxCxN(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
         t = t + 1;
      end
   end
end
t = 1;
for i = 1:2
   for j = 3:5
      for k = j+1:5
         CxNxN(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
         t = t + 1;
      end
   end
end

way2 = [CxC CxN NxN];
way3 = [CxCxN CxNxN];

X = [ X dum_x way2 way3];      % Defining Regression Matrix
```

```
% Main Effects col: 2:6

% 2-factor Interactions col: 7:16     (CxC: 7)       (CxN: 8:13)
% (NxN: 14:16)

% 3-factor Interaction col: 17:25       (CxCxN: 1719)      (CxNxN:
% 20:25)

b_part1 = regress(Y,X);    % Finding beta's

save WORKSPACE_part1;          % saving the variables for further use

normplot(b_part1);     % To plot normal plot for the regression coefficients
title(' Normal Probability Plot for Regression Coefficients');
hgsave('Normal_Plot');
figure;


% Implementing Lenth Method
bi = abs(b_part1);
s0 = 1.5*median(bi);

limit = 2.5*s0;
t = 1;
[m , n] = size(X);
for i = 1:n
   if bi(i) <= limit
      dumm(t) = bi(i);
      t = t + 1;
   end
end

PSE = 1.5*median(dumm);
ME = tinv(0.975, n/3) * PSE;
gamma = ( 1 + (0.95^(1/n))) / 2.0;

SME = tinv( gamma , n/3 ) * PSE;

hold on;

bar(b_part1);
xlabel('Effects', 'FontSize', 10, 'FontWeight', 'bold');
ylabel('Estimated Effect Contrasts', 'FontSize', 10, 'FontWeight', 'bold');

plot([0 35] , [ME ME], '--r');
plot([0 35] , [-ME -ME], '--r');

plot([0 35] , [SME SME], '-.k');
plot([0 35] , [-SME -SME], '-.k');
```

```
plot([7 7],[max(b_part1) min(b_part1)], 'k','LineWidth',2);
plot([17 17],[max(b_part1) min(b_part1)], 'k','LineWidth',2);

title('Lenth Plot ', 'FontSize', 12, 'FontWeight', 'bold');
text(7+1,max(b_part1)/1.2,'Two-Way Interactions', 'FontSize', 7);
text(17+1,max(b_part1)/1.2,'Three-Way Interactions', 'FontSize', 7);

text(30,SME,'SME');
text(30,-SME,'- SME');
text(30,ME,'ME');
text(30,-ME,'- ME');

hgsave('Lenth_Plot');
```

11.1

```
% Slider Crank Mechanism Case Study from "Generalized 3-D tolerance
% analysis of mechanical assemblies with small kinematic adjustments", Gao,
% J., Chase, K. and Magleby, S., IIE Transactions, 1998, 30, 367-377.

% We will have five variables defined by the designer. They are A, B, C, D,
% and E. There are manufacturing variations coming on to these variables.
% Together all five of them define the location of the slider with respect
% to Origin. The location of slider is U. We will take U as our response
% Variable. We will determine the order of slider crank mechanism wrt U.

% Parameters: A, B, C, D, and E

% By Jagmeet Singh on 05/27/2005

clear;clc;



% Defining Control Factor Levels.
% Control Factors are Ac, Bc, Cc, Dc, and Ec
CF_levels = [20      25;      % Ac
             12      18;      % Bc
             10      15;      % Cc
             30      40;      % Dc
             5       7];      % Ec



% Defining Control Factor Array
CF_array = ff2n(5) + 1;
[m , n] = size(CF_array);

for i = 1:m
   for j = 1:n

          % Defining Inner Array for experiments
      Inner_array(i,j) = CF_levels(j, CF_array(i,j));
   end
end



% Defining Noise Array
% Noise Factors are An, Bn, Cn, Dn, and En
NF_levels = [-0.025      0.025;      % An
```

```
        -0.0125     0.0125;     % Bn
        -0.0125     0.0125;     % Cn
        -0.03       0.03;       % Dn
        -0.0025     0.0025];     % En


% Defining Noise Array
NF_array = ff2n(5) + 1;
[m , n] = size(NF_array);

for i = 1:m
   for j = 1:n

      % Defining Outer Array Levels for the experiments
      Outer_array_levels(i,j) = NF_levels(j, NF_array(i,j));
   end
end



% Running Experiments at Inner and Outer Array Settings
[iarows, iacols] = size(Inner_array);
[oarows, oacols] =size(Outer_array_levels);
h1 = waitbar(0);


for iaruns = 1:iarows
   waitbar(iaruns/iarows, h1, sprintf('Running Control Factor Experiment # %d', iaruns));

   % Setting the Control Factor Levels for Inner Array Experiments
   Ac = Inner_array(iaruns, 1);
   Bc = Inner_array(iaruns, 2);
   Cc = Inner_array(iaruns, 3);
   Dc = Inner_array(iaruns, 4);
   Ec = Inner_array(iaruns, 5);


   index = 1;


   % Setting Outer Array Levels for Outer Array Experiments
   for i = 1:oarows
      An = Outer_array_levels(i, 1);
      Bn = Outer_array_levels(i, 2);
      Cn = Outer_array_levels(i, 3);
      Dn = Outer_array_levels(i, 4);
      En = Outer_array_levels(i, 5);
```

```
      % Defining the actual values for Parameters under a given
      % experiment
      A = Ac + An;
      B = Bc + Bn;
      C = Cc + Cn;
      D = Dc + Dn;
      E = Ec + En;


      % Calling the subroutine to find Slider Position U
      U(iaruns, index) = SliderPosition(A, B, C, D, E);


      index = index + 1;
   end
end

close(h1);
IA = CF_array*2 - 3;        % To define Inner Array in -1, 1 levels
OA = NF_array*2 - 3;        % To define Outer Array in -1, 1 levels

save WORKSPACE_1;
```

11.2

```
% Slider Crank Mechanism Case Study from "Generalized 3-D tolerance
% analysis of mechanical assemblies with small kinematic adjustments", Gao,
% J., Chase, K. and Magleby, S., IIE Transactions, 1998, 30, 367-377.

% We will have five variables defined by the designer. They are A, B, C, D,
% and E. There are manufacturing variations coming on to these variables.
% Together all five of them define the location of the slider with respect
% to Origin. The location of slider is U. We will take U as our response
% Variable.

% Parameters: A, B, C, D, and E

% To Evaluate: U

% By Jagmeet Singh on 05/27/2005

function U = SliderPosition(A, B, C, D, E)

T = (D^2)     -     ( (A - (0.7071*C) - E)^2 )     -     ( (0.7071*C)^2);

U = B + sqrt(T);

end
```

11.3

```
% Slider Crank Mechanism Case Study from "Generalized 3-D tolerance
% analysis of mechanical assemblies with small kinematic adjustments", Gao,
% J., Chase, K. and Magleby, S., IIE Transactions, 1998, 30, 367-377.

% We will have five variables defined by the designer. They are A, B, C, D,
% and E. There are manufacturing variations coming on to these variables.
% Together all five of them define the location of the slider with respect
% to Origin. The location of slider is U. We will take U as our response
% Variable. We will determine the order of slider crank mechanism wrt U.

% Parameters: A, B, C, D, and E

% By Jagmeet Singh on 05/27/2005

% INPUT: Variables from the runs of Slider Crank Position
% Solution with 2-level factors in both CF and NF array..

% OUTPUT: beta values and plotting of Normal and Lenth Plot and finding
% significant factors.

clear;clc;
load WORKSPACE_1;
clear X;

t = 1;     % Index for rows of Regression Variable

for i = 1:2^5
   for j = 1:2^5
      % Reg_Var = [CF : NF : Wr];
      % Reg_Var = [IA : OA : Wr];
      Reg_Var(t,1:5) = IA(i,:);
      Reg_Var(t,6:10) = OA(j,:);
      Reg_Var(t,11) = U(i,j);
      t = t + 1;
   end
end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ANALYSIS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% To find beta values for main effects, 2- and 3- factor interactions
Y(:,1) = Reg_Var(:,11);
X(:,1) = ones(2^5*2^5,1);
dum_x(:,1:10) = Reg_Var(:,1:10);
```

```
% Finding 2-factor interaction terms
t = 1;     % Index for storing results in dum_x
for i = 1:5
   for j = i+1:5
      CxC(:,t) = dum_x(:,i).*dum_x(:,j);
      t = t + 1;
   end
end
t = 1;
for i = 1:5
   for j = 6:10
      CxN(:,t) = dum_x(:,i).*dum_x(:,j);
      t = t + 1;
   end
end
t = 1;
for i = 6:10
   for j = i+1:10
      NxN(:,t) = dum_x(:,i).*dum_x(:,j);
      t = t + 1;
   end
end



% Finding 3-factor interactions
t = 1;
for i = 1:5
   for j = i+1:5
      for k = j+1:5
         CxCxC(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
         t = t + 1;
         sprintf('Foul Value');
      end
   end
end
t = 1;
for i = 1:5
   for j = i+1:5
      for k = 6:10
         CxCxN(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
         t = t + 1;
      end
   end
end
t = 1;
for i = 1:5
   for j = 6:10
      for k = j+1:10
         CxNxN(:,t) = dum_x(:,i).*dum_x(:,j).*dum_x(:,k);
         t = t + 1;
```

```
    end
  end
end

way2 = [CxC CxN NxN];
way3 = [CxCxC CxCxN CxNxN];

X = [ X dum_x way2 way3];      % Defining Regression Matrix

% Main Effects col: 2:11

% 2-factor Interactions col: 12:56     (CxC: 12:21)      (CxN: 22:46)
% (NxN: 47:56)

% 3-factor Interaction col: 57:166      (CxCxC: 57:66)     (CxCxN:
% 67:116)     (CxNxN: 117:166)

b_part1 = regress(Y,X);    % Finding beta's

b_part1 = b_part1(2:166);

save WORKSPACE_part1;          % saving the variables for further use

normplot(b_part1);     % To plot normal plot for the regression coefficients
title(' Normal Probability Plot for Regression Coefficients');
hgsave('Normal_Plot');
figure;


% Implementing Lenth Method
bi = abs(b_part1);
s0 = 1.5*median(bi);

limit = 2.5*s0;
t = 1;
[m , n] = size(b_part1');
for i = 1:n
   if bi(i) <= limit
      dumm(t) = bi(i);
      t = t + 1;
   end
end

PSE = 1.5*median(dumm);
ME = tinv(0.975, n/3) * PSE;
gamma = ( 1 + (0.95^(1/n))) / 2.0;

SME = tinv( gamma , n/3 ) * PSE;

hold on;
```

```
bar(b_part1);
xlabel('Effects', 'FontSize', 10, 'FontWeight', 'bold');
ylabel('Estimated Effect Contrasts', 'FontSize', 10, 'FontWeight', 'bold');

plot([0 170] , [ME ME], '--r');
plot([0 170] , [-ME -ME], '--r');

plot([0 170] , [SME SME], '-.k');
plot([0 170] , [-SME -SME], '-.k');

plot([10 10],[max(b_part1) min(b_part1)], 'k','LineWidth',2);
plot([55 55],[max(b_part1) min(b_part1)], 'k','LineWidth',2);

title('Lenth Plot ', 'FontSize', 12, 'FontWeight', 'bold');
text(10+1,max(b_part1)/1.2,'Two-Way Interactions', 'FontSize', 7);
text(55+1,max(b_part1)/1.2,'Three-Way Interactions', 'FontSize', 7);

text(167,SME,'SME');
text(167,-SME,'- SME');
text(167,ME,'ME');
text(167,-ME,'- ME');

hgsave('Lenth_Plot');
```

12.1

Working with Hierarchical Probability Model, Weak Hierarchy for Compound Noise

$$Table1 := \begin{pmatrix} 10 & 1 & 1 & 1 & 1 \\ 10 & 1 & 1 & 0.1 & 0.1 \\ 10 & 1 & 0 & 1 & 1 \\ 15 & \frac{1}{3} & \frac{2}{3} & 1 & 1 \\ 15 & \frac{1}{3} & \frac{2}{3} & 0.1 & 0.1 \\ 15 & \frac{1}{3} & 0 & 1 & 1 \end{pmatrix}$$

$$Table2 := \begin{pmatrix} 0.25 & 0.25 & 0.1 & 0 & 0.25 & 0.1 & 0 & 0 \\ 0.25 & 0.25 & 0.1 & 0 & 0.25 & 0.1 & 0 & 0 \\ 0.25 & 0.25 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0.43 & 0.31 & 0.04 & 0 & 0.17 & 0.08 & 0.02 & 0 \\ 0.43 & 0.31 & 0.04 & 0 & 0.17 & 0.08 & 0.02 & 0 \\ 0.43 & 0.31 & 0.04 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Table := augment(Table1, Table2)

Defining the Parameters for Hierarchical Probability Model.

ncf := 7 Number of Control Factors

nnf := 5 Number of Noise Factors

total_var := nnf + ncf

Model_to_be_used := 1

1 for Basic WH
2 for Basic low *w*
3 for Basic 2nd order
4 for Fitted WH
5 for Fitted low *w*
6 for Fitted 2nd order

i := Model_to_be_used − 1

j := i

Defining Parameter of the Model

c := Table$_{0,0}$ s1 := Table$_{1,1}$ s2 := Table$_{2,2}$ w1 := Table$_{3,3}$ w2 := Table$_{4,4}$

p := Table$_{5,5}$ p11 := Table$_{6,6}$ p01 := Table$_{7,7}$ p00 := Table$_{8,8}$ p111 := Table$_{9,9}$

p011 := Table$_{10,10}$ p001 := Table$_{11,11}$ p000 := Table$_{12,12}$

Number of Experiments to be peformed

max_expts := 500 expt := 0 .. (max_expts − 1)

$i := total\_var - 1$ $j := i$ $k := j$

$\delta_{expt} := runif(ncf + nnf, 0, 1)$

$$\delta_{expt} := \left| \begin{array}{l} D \leftarrow 0 \\ \text{for } i \in 0..ncf + nnf - 1 \\ \quad \left| \begin{array}{l} D_{i,0} \leftarrow 0 \text{ if } (\delta_{expt})_{i,0} > p \\ D_{i,0} \leftarrow 1 \text{ otherwise} \end{array} \right. \\ D \end{array} \right.$$

For active main effects

$$\delta ij_{expt} := \left| \begin{array}{l} D \leftarrow 0 \\ \text{for } i \in 0..(total\_var - 2) \\ \quad \text{for } j \in (i+1)..(total\_var - 1) \\ \quad \left| \begin{array}{l} sum\_\delta \leftarrow (\delta_{expt})_{i,0} + (\delta_{expt})_{j,0} \\ dummy\_\delta \leftarrow runif(1,0,1) \\ D_{i,j} \leftarrow 1 \text{ if } \left[(sum\_\delta = 0) \wedge (dummy\_\delta_{0,0} \le p00) \right] \\ D_{i,j} \leftarrow 0 \text{ if } \left[(sum\_\delta = 0) \wedge (dummy\_\delta_{0,0} > p00) \right] \\ D_{i,j} \leftarrow 1 \text{ if } \left[(sum\_\delta = 1) \wedge (dummy\_\delta_{0,0} \le p01) \right] \\ D_{i,j} \leftarrow 0 \text{ if } \left[(sum\_\delta = 0) \wedge (dummy\_\delta_{0,0} > p01) \right] \\ D_{i,j} \leftarrow 1 \text{ if } \left[(sum\_\delta = 2) \wedge (dummy\_\delta_{0,0} \le p11) \right] \\ D_{i,j} \leftarrow 0 \text{ if } \left[(sum\_\delta = 2) \wedge (dummy\_\delta_{0,0} > p11) \right] \end{array} \right. \\ D \end{array} \right.$$

For active two-factor interactions

$$zeros := \left| \begin{array}{l} D \leftarrow 0 \\ \text{for } i \in 0..k \\ \quad D_{i,0} \leftarrow 0 \\ D \end{array} \right.$$

$$\delta ijk_{expt} := \left| \begin{array}{l} D \leftarrow 0 \\ \text{for } i \in 0..k \\ \quad D_{i,0} \leftarrow 0 \\ \text{for } i \in 0..k \\ \quad \text{for } j \in 0..k \\ \quad (\delta ijk_{expt})_{i,j} \leftarrow D \\ \delta ijk_{expt} \end{array} \right.$$

$\delta ijk_{expt} :=$

$D \leftarrow 0$

$A \leftarrow 0$

for $i \in 0..k$

$D_{i,0} \leftarrow 0$

for $i \in 0..k$

for $j \in 0..k$

$(\delta ijk_{expt})_{i,j} \leftarrow D$

For active three-factor interactions

for $i \in 0..(total\_var - 3)$

for $j \in (i+1)..(total\_var - 2)$

for $k \in (j+1)..(total\_var - 1)$

$sum\_\delta \leftarrow (\delta_{expt})_{i,0} + (\delta_{expt})_{j,0} + (\delta_{expt})_{k,0}$

$dummy\_\delta \leftarrow runif(1,0,1)$

$A \leftarrow 1 \text{ if } [(sum\_\delta = 0) \wedge (dummy\_\delta_{0,0} \le p000)]$

$A \leftarrow 0 \text{ if } [(sum\_\delta = 0) \wedge (dummy\_\delta_{0,0} > p000)]$

$A \leftarrow 1 \text{ if } [(sum\_\delta = 1) \wedge (dummy\_\delta_{0,0} \le p001)]$

$A \leftarrow 0 \text{ if } [(sum\_\delta = 1) \wedge (dummy\_\delta_{0,0} > p001)]$

$A \leftarrow 1 \text{ if } [(sum\_\delta = 2) \wedge (dummy\_\delta_{0,0} \le p011)]$

$A \leftarrow 0 \text{ if } [(sum\_\delta = 2) \wedge (dummy\_\delta_{0,0} > p011)]$

$A \leftarrow 1 \text{ if } [(sum\_\delta = 3) \wedge (dummy\_\delta_{0,0} \le p111)]$

$A \leftarrow 0 \text{ if } [(sum\_\delta = 3) \wedge (dummy\_\delta_{0,0} > p111)]$

$[(\delta ijk_{expt})_{i,j}]_{k,0} \leftarrow A$

δijk_{expt}

$t :=$

$t \leftarrow 0$

for $i \in 0..k$

$t_i \leftarrow w1 \text{ if } [i \le (nnf - 1)]$

$t_i \leftarrow 1 \text{ otherwise}$

t

For Main Effect Coefficients

$$\beta i_{expt} := \left| \begin{array}{l} \beta i_{expt} \leftarrow 0 \\ \text{for } i \in 0..k \\ \quad \left| \begin{array}{l} \left(\beta i_{expt}\right)_i \leftarrow t_i \cdot \text{rnorm}(1,0,1)_{0,0} \text{ if } \left(\delta_{expt}\right)_i = 0 \\ \left(\beta i_{expt}\right)_i \leftarrow t_i \cdot \text{rnorm}(1,0,c)_{0,0} \text{ otherwise} \end{array} \right. \\ \left(\beta i_{expt}\right) \end{array} \right.$$

For Two-Way Interaction Effects

$$\beta ij_{expt} := \left| \begin{array}{l} \beta ij_{expt} \leftarrow 0 \\ \text{for } i \in 0..k \\ \quad \text{for } j \in 0..k \\ \quad \left(\beta ij_{expt}\right)_{i,j} \leftarrow 0 \\ \text{for } i \in 0..(\text{total\_var} - 3) \\ \quad \text{for } j \in (i+1)..(\text{total\_var} - 2) \\ \quad \left| \begin{array}{l} \left(\beta ij_{expt}\right)_{i,j} \leftarrow \left[\left(t_i\right) \cdot t_j \cdot \text{rnorm}(1,0,s1)_{0,0}\right] \text{ if } \left(\delta ij_{expt}\right)_{i,j} = 0 \\ \left(\beta ij_{expt}\right)_{i,j} \leftarrow \left[\left(t_i\right) \cdot t_j \cdot \text{rnorm}(1,0,c \cdot s1)_{0,0}\right] \text{ otherwise} \end{array} \right. \\ \beta ij_{expt} \end{array} \right.$$

For Three-Way Interaction Effects

$$\beta ijk_{expt} := \left| \begin{array}{l} \beta ijk_{expt} \leftarrow 0 \\ D \leftarrow 0 \\ \text{for } i \in 0..k \\ \quad D_{i,0} \leftarrow 0 \\ \text{for } i \in 0..k \\ \quad \text{for } j \in 0..k \\ \quad \left(\beta ijk_{expt}\right)_{i,j} \leftarrow D \\ \text{for } i \in 0..(\text{total\_var} - 3) \\ \quad \text{for } j \in (i+1)..(\text{total\_var} - 2) \\ \quad \text{for } k \in (j+1)..(\text{total\_var} - 1) \\ \quad \left| \begin{array}{l} \left[\left(\beta ijk_{expt}\right)_{i,j}\right]_{k,0} \leftarrow \left(t_i \cdot t_j \cdot t_k \cdot \text{rnorm}(1,0,s2)_{0,0}\right) \text{ if } \left[\left[\left(\delta ijk_{expt}\right)_{i,j}\right]_{k,0}\right] = 0 \\ \left[\left(\beta ijk_{expt}\right)_{i,j}\right]_{k,0} \leftarrow \left(t_i \cdot t_j \cdot t_k \cdot \text{rnorm}(1,0,c \cdot s2)_{0,0}\right) \text{ otherwise} \end{array} \right. \\ \beta ijk_{expt} \end{array} \right.$$

Control Factor Array

CF_array := stack(CF_array1, CF_array2)

NF_array :=

−1	−1	−1	−1	−1
−1	−1	−1	−1	1
−1	−1	−1	1	−1
−1	−1	−1	1	1
−1	−1	1	−1	−1
−1	−1	1	−1	1
−1	−1	1	1	−1
−1	−1	1	1	1
−1	1	−1	−1	−1
−1	1	−1	−1	1
−1	1	−1	1	−1
−1	1	−1	1	1
−1	1	1	−1	−1
−1	1	1	−1	1
−1	1	1	1	−1
−1	1	1	1	1
1	−1	−1	−1	−1
1	−1	−1	−1	1
1	−1	−1	1	−1
1	−1	−1	1	1
1	−1	1	−1	−1
1	−1	1	−1	1
1	−1	1	1	−1
1	−1	1	1	1
1	1	−1	−1	−1
1	1	−1	−1	1
1	1	−1	1	−1
1	1	−1	1	1
1	1	1	−1	−1
1	1	1	−1	1
1	1	1	1	−1
1	1	1	1	1

Noise Factor Array

$$
\begin{aligned}
&y_{expt} := \left| \begin{array}{l}
y_{expt} \leftarrow 0 \\
\text{for } cfruns \in 0..(128-1) \\
\quad \text{for } nfruns \in 0..(32-1) \\
\quad\quad x1_{expt} \leftarrow \left(CF\_array^{T}\right)^{\langle cfruns \rangle} \\
\quad\quad x2_{expt} \leftarrow \left(NF\_array^{T}\right)^{\langle nfruns \rangle} \\
\quad\quad x_{expt} \leftarrow augment\left(x2_{expt}^{T}, x1_{expt}^{T}\right) \\
\quad\quad sum1_{expt} \leftarrow \sum_{i=0}^{11} \left[\left[\left(\beta i_{expt}\right)_{i}\right] \cdot \left(x_{expt}\right)_{0,i}\right] \\
\quad\quad sum2_{expt} \leftarrow \sum_{i=0}^{11} \sum_{j=0}^{11} \left[\left[\left[\left(\beta ij_{expt}\right)_{i,j}\right] \cdot \left(x_{expt}\right)_{0,i}\right] \cdot \left(x_{expt}\right)_{0,j}\right] \\
\quad\quad sum3_{expt} \leftarrow \sum_{i=0}^{11} \sum_{j=0}^{11} \sum_{k=0}^{11} \left[\left[\left(\beta ijk_{expt}\right)_{i,j}\right]_{k,0} \cdot \left(x_{expt}\right)_{0,i} \cdot \left(x_{expt}\right)_{0,j} \cdot \left(x_{expt}\right)_{0,k}\right] \\
\quad\quad \left(y_{expt}\right)_{cfruns,nfruns} \leftarrow sum1_{expt} + sum2_{expt} + sum3_{expt} + rnorm(1,0,w2)_{0,0} \\
y_{expt}
\end{array} \right.
\end{aligned}
$$

Finding the Variance of Response at each CF Setting and finding the robust setting

$$\text{Variance\_y}_{expt} := \left| \begin{array}{l} \text{for } cfruns \in 0 .. (128 - 1) \\ \quad \left| \begin{array}{l} \text{yrows}_{expt} \leftarrow \left(y_{expt}^{T}\right)^{\langle cfruns \rangle} \\ \left(\text{Variance\_y}_{expt}\right)_{cfruns} \leftarrow \text{var}\left(\text{yrows}_{expt}\right) \end{array} \right. \\ \text{Variance\_y}_{expt} \end{array} \right.$$

$$\text{Augmented\_Matrix}_{expt} := \text{augmen}\left(\text{Variance\_y}_{expt}, \text{CF\_array}\right)$$

$$\text{Actual\_Robust\_Setting}_{expt} := \text{csort}\left(\text{Augmented\_Matrix}_{expt}, 0\right)$$

$$\text{Robust\_Setting}_{expt} := \left(\text{Actual\_Robust\_Setting}_{expt}^{T}\right)^{\langle 0 \rangle}$$

$$\text{Minimum\_Variance}_{expt} := \left(\text{Robust\_Setting}_{expt}\right)_{0,0}$$

$$\text{Average\_Variance}_{expt} := \text{mean}\left(\text{Variance\_y}_{expt}\right)$$

$$\text{Robust\_Setting}_{expt} := \left| \begin{array}{l} \text{for } i \in 1 .. 7 \\ \quad (A)_{(i-1),0} \leftarrow \left(\text{Robust\_Setting}_{expt}\right)_{i,0} \\ A \end{array} \right.$$

$$\text{Robust\_Setting} = \begin{pmatrix} -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$

Using Compound Noise to Predict the Robust Setting

Setting up Compound Noise (Extreme)

$$\text{CN\_setting}_{expt} := \left| \begin{array}{l} \text{for } i \in 0..(nnf-1) \\ \quad \left| \begin{array}{l} A_{i,0} \leftarrow \text{sign}\left[\left(\beta i_{expt}\right)_i\right] \\ A_{i,1} \leftarrow -1 \cdot \text{sign}\left[\left(\beta i_{expt}\right)_i\right] \end{array} \right. \\ A \end{array} \right.$$

$$\text{CN\_setting} = \begin{pmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{pmatrix}$$

Finding Response Under Compound Noise

$$\text{y\_comp}_{expt} := \left| \begin{array}{l} \text{y\_comp}_{expt} \\ \text{for cfruns} \in 0..(128-1) \\ \quad \text{for nfruns} \in 0..(2-1) \\ \quad \left| \begin{array}{l} x1_{expt} \leftarrow \left(\text{CF\_array}^T\right)^{\langle \text{cfruns} \rangle} \\ x2_{expt} \leftarrow \left(\text{CN\_setting}_{expt}\right)^{\langle \text{nfruns} \rangle} \\ x_{expt} \leftarrow \text{augment}\left[\left(x2_{expt}^T\right), x1_{expt}^T\right] \\ \text{sum}1_{expt} \leftarrow \sum_{i=0}^{11} \left[\left[\left(\beta i_{expt}\right)_i\right] \cdot \left(x_{expt}\right)_{0,i}\right] \\ \text{sum}2_{expt} \leftarrow \sum_{i=0}^{11} \sum_{j=0}^{11} \left[\left[\left[\left(\beta ij_{expt}\right)_{i,j}\right] \cdot \left(x_{expt}\right)_{0,i}\right] \cdot \left(x_{expt}\right)_{0,j}\right] \\ \text{sum}3_{expt} \leftarrow \sum_{i=0}^{11} \sum_{j=0}^{11} \sum_{k=0}^{11} \left[\left[\left[\left(\beta ijk_{expt}\right)_{i,j}\right]_{k,0} \cdot \left(x_{expt}\right)_{0,i} \cdot \left(x_{expt}\right)_{0,j} \cdot \left(x_{expt}\right)_{0,k}\right] \\ \left(\text{y\_comp}_{expt}\right)_{\text{cfruns},\text{nfruns}} \leftarrow \text{sum}1_{expt} + \text{sum}2_{expt} + \text{sum}3_{expt} + \text{rnorm}(1,0,w2)_{0,0} \end{array} \right. \\ \text{y\_comp}_{expt} \end{array} \right.$$

Finding the Variance of Response at each CF setting under Compound Noise and Predict the Robust Setting

$$\text{Variance\_y\_comp}_{expt} := \left| \begin{array}{l} \text{for } cfruns \in 0..(128-1) \\ \quad \left| \begin{array}{l} \text{yrows}_{expt} \leftarrow \left(\text{y\_comp}_{expt}^{T}\right)^{\langle cfruns \rangle} \\ \left(\text{Variance\_y\_comp}_{expt}\right)_{cfruns} \leftarrow \text{var}\left(\text{yrows}_{expt}\right) \end{array} \right. \\ \text{Variance\_y\_comp}_{expt} \end{array} \right.$$

$$\text{Augmented\_Matrix\_comp}_{expt} := \text{augment}\left(\text{Variance\_y\_comp}_{expt}, \text{CF\_array}\right)$$

$$\text{Predicted\_Robust\_Setting}_{expt} := \text{csort}\left(\text{Augmented\_Matrix\_comp}_{expt}, 0\right)$$

$$\text{Robust\_Setting\_comp}_{expt} := \left(\text{Predicted\_Robust\_Setting}_{expt}^{T}\right)^{\langle 0 \rangle}$$

$$\text{Minimum\_Variance\_comp}_{expt} := \left(\text{Robust\_Setting\_comp}_{expt}\right)_{0,0}$$

$$\text{Robust\_Setting\_comp}_{expt} := \left| \begin{array}{l} \text{for } i \in 1..7 \\ \quad (A)_{(i-1),0} \leftarrow \left(\text{Robust\_Setting\_comp}_{expt}\right)_{i,0} \\ A \end{array} \right.$$

$$\text{Robust\_Setting}_2 = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \end{pmatrix} \qquad \text{Robust\_Setting\_comp}_2 = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$$

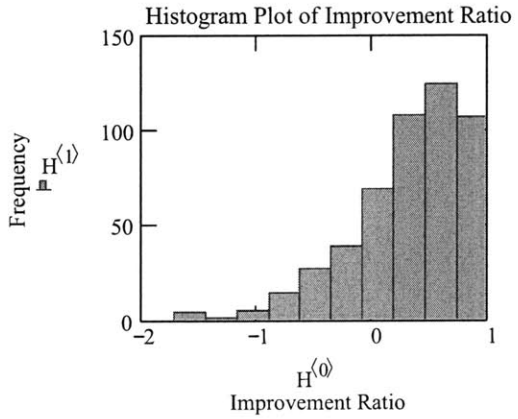
Finding the Variance at Predicted Robust Setting by Compound Noise

$$\text{Predicted\_Variance}_{expt} := \left| \begin{array}{l} \text{for } cfruns \in 0..(128-1) \\ \quad \left| \begin{array}{l} setting_{expt} \leftarrow \left| \begin{array}{l} \text{for } i \in 1..7 \\ \quad A_{(i-1),0} \leftarrow \left[\left(\text{Actual\_Robust\_Setting}_{expt}^{T} \right)^{\langle cfruns \rangle} \right]_{i,0} \\ A \end{array} \right. \\ Var \leftarrow \left[\left(\text{Actual\_Robust\_Setting}_{expt}^{T} \right)^{\langle cfruns \rangle} \right]_{0,0} \\ \text{break if } \left(setting_{expt} = \text{Robust\_Setting\_comp}_{expt} \right) \end{array} \right. \\ Var \end{array} \right.$$

Defining Improvement Ratio

$$\text{Improvement\_Ratio}_{expt} := \frac{\left(\text{Average\_Variance}_{expt} - \text{Predicted\_Variance}_{expt} \right)}{\left(\text{Average\_Variance}_{expt} - \text{Minimum\_Variance}_{expt} \right)}$$

H := histogram(10, Improvement_Rati)



mean(Improvement_Rati) = 0.313

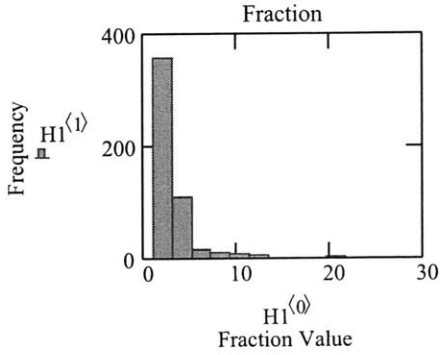
median(Improvement_Rati) = 0.407

Overall Results

Defining Ratio of Predicted Variance to Minimum Variance

$$\text{Fraction1}_{expt} := \frac{\text{Predicted\_Variance}_{expt}}{\text{Minimum\_Variance}_{expt}}$$

H1 := histogram(10, Fraction1)



mean(Fraction1) = 2.817

median(Fraction1) = 2.243

13.1

Working with Hierarchical Probability Model, Strong Hierarchy Model for Compound Noise

$$\text{Table2} := \begin{pmatrix} 0.25 & 0.25 & 0.1 & 0 & 0.25 & 0.1 & 0 & 0 \\ 0.25 & 0.25 & 0.1 & 0 & 0.25 & 0.1 & 0 & 0 \\ 0.25 & 0.25 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0.43 & 0.31 & 0.04 & 0 & 0.17 & 0.08 & 0.02 & 0 \\ 0.43 & 0.31 & 0.04 & 0 & 0.17 & 0.08 & 0.02 & 0 \\ 0.43 & 0.31 & 0.04 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Table := augment(Table1, Table2)

Defining the Parameters for Hierarchical Probability Model.

ncf := 7 Number of Control Factors

nnf := 5 Number of Noise Factors

total_var := nnf + ncf

Model_to_be_used := 3

1 for Basic WH
2 for Basic low *w*
3 for Basic 2nd order
4 for Fitted WH
5 for Fitted low *w*
6 for Fitted 2nd order

i := Model_to_be_used − 1

j := i

Defining Parameter of the Model

c := Table$_{1,0}$ s1 := Table$_{1,1}$ s2 := Table$_{1,2}$ w1 := Table$_{1,3}$ w2 := Table$_{1,4}$

p := Table$_{1,5}$ p11 := Table$_{1,6}$ p01 := Table$_{1,7}$ p00 := Table$_{1,8}$ p111 := Table$_{1,9}$

p011 := Table$_{1,10}$ p001 := Table$_{1,11}$ p000 := Table$_{1,12}$

Number of Experiments to be peformed

max_expts := 500 expt := 0 .. (max_expts − 1)

$$\text{Table1} := \begin{pmatrix} 10 & 1 & 1 & 1 & 1 \\ 10 & 1 & 1 & 0.1 & 0.1 \\ 10 & 1 & 0 & 1 & 1 \\ 15 & \frac{1}{3} & \frac{2}{3} & 1 & 1 \\ 15 & \frac{1}{3} & \frac{2}{3} & 0.1 & 0.1 \\ 15 & \frac{1}{3} & 0 & 1 & 1 \end{pmatrix}$$

$i := total\_var - 1$ $j := i$ $k := j$

$\delta_{expt} := runif(ncf + nnf, 0, 1)$

$\delta_{expt} :=$
$D \leftarrow 0$
for $i \in 0 .. ncf + nnf - 1$
$D_{i,0} \leftarrow 0$ if $(\delta_{expt})_{i,0} > p$
$D_{i,0} \leftarrow 1$ otherwise
D

For active main effects

$(\delta ij_{expt})_{i,j} := 0$

$\delta ij_{expt} :=$
$D \leftarrow 0$
for $i \in 0 .. (total\_var - 2)$
for $j \in (i + 1) .. (total\_var - 1)$
$sum\_\delta \leftarrow (\delta_{expt})_{i,0} + (\delta_{expt})_{j,0}$
$dummy\_\delta \leftarrow runif(1, 0, 1)$
$D_{i,j} \leftarrow 1$ if $[(sum\_\delta = 0) \wedge (dummy\_\delta_{0,0} \le p00)]$
$D_{i,j} \leftarrow 0$ if $[(sum\_\delta = 0) \wedge (dummy\_\delta_{0,0} > p00)]$
$D_{i,j} \leftarrow 1$ if $[(sum\_\delta = 1) \wedge (dummy\_\delta_{0,0} \le p01)]$
$D_{i,j} \leftarrow 0$ if $[(sum\_\delta = 0) \wedge (dummy\_\delta_{0,0} > p01)]$
$D_{i,j} \leftarrow 1$ if $[(sum\_\delta = 2) \wedge (dummy\_\delta_{0,0} \le p11)]$
$D_{i,j} \leftarrow 0$ if $[(sum\_\delta = 2) \wedge (dummy\_\delta_{0,0} > p11)]$
D

For active two-factor interactions

$zeros :=$
$D \leftarrow 0$
for $i \in 0 .. k$
$D_{i,0} \leftarrow 0$
D

$\delta ijk_{expt} :=$
$D \leftarrow 0$
for $i \in 0 .. k$
$D_{i,0} \leftarrow 0$
for $i \in 0 .. k$
for $j \in 0 .. k$
$(\delta ijk_{expt})_{i,j} \leftarrow D$
δijk_{expt}

For active three-factor interactions

```
δijk_expt :=
  D ← 0
  A ← 0
  for i ∈ 0..k
    D_{i,0} ← 0
  for i ∈ 0..k
    for j ∈ 0..k
      (δijk_expt)_{i,j} ← D
  for i ∈ 0..(total_var − 3)
    for j ∈ (i + 1)..(total_var − 2)
      for k ∈ (j + 1)..(total_var − 1)
        sum_δ ← (δ_expt)_{i,0} + (δ_expt)_{j,0} + (δ_expt)_{k,0}
        dummy_δ ← runif(1, 0, 1)
        A ← 1  if [(sum_δ = 0) ∧ (dummy_δ_{0,0} ≤ p000)]
        A ← 0  if [(sum_δ = 0) ∧ (dummy_δ_{0,0} > p000)]
        A ← 1  if [(sum_δ = 1) ∧ (dummy_δ_{0,0} ≤ p001)]
        A ← 0  if [(sum_δ = 1) ∧ (dummy_δ_{0,0} > p001)]
        A ← 1  if [(sum_δ = 2) ∧ (dummy_δ_{0,0} ≤ p011)]
        A ← 0  if [(sum_δ = 2) ∧ (dummy_δ_{0,0} > p011)]
        A ← 1  if [(sum_δ = 3) ∧ (dummy_δ_{0,0} ≤ p111)]
        A ← 0  if [(sum_δ = 3) ∧ (dummy_δ_{0,0} > p111)]
        [(δijk_expt)_{i,j}]_{k,0} ← A
  δijk_expt
```

```
t :=
  t ← 0
  for i ∈ 0..k
    t_i ← w1  if [i ≤ (nnf − 1)]
    t_i ← 1  otherwise
  t
```


For Main Effect Coefficients

$\beta i_{expt} :=$ | $\beta i_{expt} \leftarrow 0$
for $i \in 0..k$
$(\beta i_{expt})_i \leftarrow t_i \cdot rnorm(1,0,1)_{0,0}$ if $(\delta_{expt})_i = 0$
$(\beta i_{expt})_i \leftarrow t_i \cdot rnorm(1,0,c)_{0,0}$ otherwise
(βi_{expt})

$c = 10$

$total\_var = 12$

For Two-Way Interaction Effects

$\beta ij_{expt} :=$ | $\beta ij_{expt} \leftarrow 0$
for $i \in 0..k$
for $j \in 0..k$
$(\beta ij_{expt})_{i,j} \leftarrow 0$
for $i \in 0..(total\_var - 3)$
for $j \in (i+1)..(total\_var - 2)$
$(\beta ij_{expt})_{i,j} \leftarrow [(t_i) \cdot t_j \cdot rnorm(1,0,s1)_{0,0}]$ if $(\delta ij_{expt})_{i,j} = 0$
$(\beta ij_{expt})_{i,j} \leftarrow [(t_i) \cdot t_j \cdot rnorm(1,0,c \cdot s1)_{0,0}]$ otherwise
βij_{expt}

$s2 := 0.00000000001$

For Three-Way Interaction Effects

$\beta ijk_{expt} :=$ | $\beta ijk_{expt} \leftarrow 0$
$D \leftarrow 0$
for $i \in 0..k$
$D_{i,0} \leftarrow 0$
for $i \in 0..k$
for $j \in 0..k$
$(\beta ijk_{expt})_{i,j} \leftarrow D$
for $i \in 0..(total\_var - 3)$
for $j \in (i+1)..(total\_var - 2)$
for $k \in (j+1)..(total\_var - 1)$
$[(\beta ijk_{expt})_{i,j}]_{k,0} \leftarrow (t_i \cdot t_j \cdot t_k \cdot rnorm(1,0,s2)_{0,0})$ if $[[(\delta ijk_{expt})_{i,j}]_{k,0}] = 0$
$[(\beta ijk_{expt})_{i,j}]_{k,0} \leftarrow (t_i \cdot t_j \cdot t_k \cdot rnorm(1,0,c \cdot s2)_{0,0})$ otherwise
βijk_{expt}

Control Factor Array

$s2 = 1 \times 10^{-11}$

CF_array := stack(CF_array1, CF_array2)

NF_array :=

$$\begin{pmatrix}
-1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 \\
-1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & 1 & 1 \\
-1 & -1 & 1 & -1 & -1 \\
-1 & -1 & 1 & -1 & 1 \\
-1 & -1 & 1 & 1 & -1 \\
-1 & -1 & 1 & 1 & 1 \\
-1 & 1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & 1 \\
-1 & 1 & -1 & 1 & -1 \\
-1 & 1 & -1 & 1 & 1 \\
-1 & 1 & 1 & -1 & -1 \\
-1 & 1 & 1 & -1 & 1 \\
-1 & 1 & 1 & 1 & -1 \\
-1 & 1 & 1 & 1 & 1 \\
1 & -1 & -1 & -1 & -1 \\
1 & -1 & -1 & -1 & 1 \\
1 & -1 & -1 & 1 & -1 \\
1 & -1 & -1 & 1 & 1 \\
1 & -1 & 1 & -1 & -1 \\
1 & -1 & 1 & -1 & 1 \\
1 & -1 & 1 & 1 & -1 \\
1 & -1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & 1 \\
1 & 1 & -1 & 1 & -1 \\
1 & 1 & -1 & 1 & 1 \\
1 & 1 & 1 & -1 & -1 \\
1 & 1 & 1 & -1 & 1 \\
1 & 1 & 1 & 1 & -1 \\
1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

Noise Factor Array

$y_{expt} :=$ | $y_{expt} \leftarrow 0$

for $cfruns \in 0..(128-1)$

for $nfruns \in 0..(32-1)$

$x1_{expt} \leftarrow \left(CF\_array^{T}\right)^{\langle cfruns \rangle}$

$x2_{expt} \leftarrow \left(NF\_array^{T}\right)^{\langle nfruns \rangle}$

$x_{expt} \leftarrow augment\left(x2_{expt}^{T}, x1_{expt}^{T}\right)$

$$sum1_{expt} \leftarrow \sum_{i=0}^{11} \left[\left[\left(\beta i_{expt}\right)_{i}\right] \cdot \left(x_{expt}\right)_{0,i}\right]$$

$$sum2_{expt} \leftarrow \sum_{i=0}^{11}\sum_{j=0}^{11} \left[\left[\left[\left(\beta ij_{expt}\right)_{i,j}\right] \cdot \left(x_{expt}\right)_{0,i}\right] \cdot \left(x_{expt}\right)_{0,j}\right]$$

$$sum3_{expt} \leftarrow \sum_{i=0}^{11}\sum_{j=0}^{11}\sum_{k=0}^{11} \left[\left[\left(\beta ijk_{expt}\right)_{i,j}\right]_{k,0} \cdot \left(x_{expt}\right)_{0,i} \cdot \left(x_{expt}\right)_{0,j} \cdot \left(x_{expt}\right)_{0,k}\right]$$

$$\left(y_{expt}\right)_{cfruns,nfruns} \leftarrow sum1_{expt} + sum2_{expt} + sum3_{expt} + rnorm(1,0,w2)_{0,0}$$

y_{expt}

Finding the Variance of Response at each CF Setting and finding the robust setting

$$\text{Variance\_y}_{expt} := \left| \begin{array}{l} \text{for } cfruns \in 0..(128-1) \\ \quad \left| \begin{array}{l} \text{yrows}_{expt} \leftarrow \left(y_{expt}^{T}\right)^{\langle cfruns \rangle} \\ \left(\text{Variance\_y}_{expt}\right)_{cfruns} \leftarrow \text{var}\left(\text{yrows}_{expt}\right) \end{array} \right. \\ \text{Variance\_y}_{expt} \end{array} \right.$$

$$\text{Augmented\_Matrix}_{expt} := \text{augment}\left(\text{Variance\_y}_{expt}, \text{CF\_array}\right)$$

$$\text{Actual\_Robust\_Setting}_{expt} := \text{csort}\left(\text{Augmented\_Matrix}_{expt}, 0\right)$$

$$\text{Robust\_Setting}_{expt} := \left(\text{Actual\_Robust\_Setting}_{expt}^{T}\right)^{\langle 0 \rangle}$$

$$\text{Minimum\_Variance}_{expt} := \left(\text{Robust\_Setting}_{expt}\right)_{0,0}$$

$$\text{Average\_Variance}_{expt} := \text{mean}\left(\text{Variance\_y}_{expt}\right)$$

$$\text{Robust\_Setting}_{expt} := \left| \begin{array}{l} \text{for } i \in 1..7 \\ \quad (A)_{(i-1),0} \leftarrow \left(\text{Robust\_Setting}_{expt}\right)_{i,0} \\ A \end{array} \right.$$

$$\text{Robust\_Setting}_g = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

Using Compound Noise to Predict the Robust Setting

Setting up Compound Noise (Extreme)

$$\text{CN\_setting}_{expt} := \left| \begin{array}{l} \text{for } i \in 0..(nnf-1) \\ \quad A_{i,0} \leftarrow \text{sign}\left[\left(\beta i_{expt}\right)_i\right] \\ \quad A_{i,1} \leftarrow -1 \cdot \text{sign}\left[\left(\beta i_{expt}\right)_i\right] \\ A \end{array} \right.$$

$$\text{CN\_setting}_g = \begin{pmatrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \end{pmatrix}$$

Finding Response Under Compound Noise

$$\text{y\_comp}_{expt} := \left| \begin{array}{l} \text{y\_comp}_{expt} \\ \text{for } cfruns \in 0..(128-1) \\ \quad \text{for } nfruns \in 0..(2-1) \\ \quad\quad x1_{expt} \leftarrow \left(\text{CF\_array}^T\right)^{\langle cfruns \rangle} \\ \quad\quad x2_{expt} \leftarrow \left(\text{CN\_setting}_{expt}\right)^{\langle nfruns \rangle} \\ \quad\quad x_{expt} \leftarrow \text{augment}\left[\left(x2_{expt}^T\right), x1_{expt}^T\right] \\ \quad\quad sum1_{expt} \leftarrow \sum_{i=0}^{11} \left[\left[\left(\beta i_{expt}\right)_i\right] \cdot \left(x_{expt}\right)_{0,i}\right] \\ \quad\quad sum2_{expt} \leftarrow \sum_{i=0}^{11} \sum_{j=0}^{11} \left[\left[\left[\left(\beta ij_{expt}\right)_{i,j}\right] \cdot \left(x_{expt}\right)_{0,i}\right] \cdot \left(x_{expt}\right)_{0,j}\right] \\ \quad\quad sum3_{expt} \leftarrow \sum_{i=0}^{11} \sum_{j=0}^{11} \sum_{k=0}^{11} \left[\left[\left(\beta ijk_{expt}\right)_{i,j}\right]_{k,0} \cdot \left(x_{expt}\right)_{0,i} \cdot \left(x_{expt}\right)_{0,j} \cdot \left(x_{expt}\right)_{0,k}\right] \\ \quad\quad \left(\text{y\_comp}_{expt}\right)_{cfruns,nfruns} \leftarrow sum1_{expt} + sum2_{expt} + sum3_{expt} + \text{rnorm}(1,0,w2)_{0,0} \\ \text{y\_comp}_{expt} \end{array} \right.$$

Finding the Variance of Response at each CF setting under Compound Noise and Predict the Robust Setting

$$\text{Variance\_y\_comp}_{expt} := \left| \begin{array}{l} \text{for } cfruns \in 0..(128-1) \\ \quad \left| \begin{array}{l} \text{yrows}_{expt} \leftarrow \left(\text{y\_comp}_{expt}^{T}\right)^{\langle cfruns \rangle} \\ \left(\text{Variance\_y\_comp}_{expt}\right)_{cfruns} \leftarrow \text{var}\left(\text{yrows}_{expt}\right) \end{array} \right. \\ \text{Variance\_y\_comp}_{expt} \end{array} \right.$$

$$\text{Augmented\_Matrix\_comp}_{expt} := \text{augment}\left(\text{Variance\_y\_comp}_{expt}, \text{CF\_array}\right)$$

$$\text{Predicted\_Robust\_Setting}_{expt} := \text{csort}\left(\text{Augmented\_Matrix\_comp}_{expt}, 0\right)$$

$$\text{Robust\_Setting\_comp}_{expt} := \left(\text{Predicted\_Robust\_Setting}_{expt}^{T}\right)^{\langle 0 \rangle}$$

$$\text{Minimum\_Variance\_comp}_{expt} := \left(\text{Robust\_Setting\_comp}_{expt}\right)_{0,0}$$

$$\text{Robust\_Setting\_comp}_{expt} := \left| \begin{array}{l} \text{for } i \in 1..7 \\ \quad (A)_{(i-1),0} \leftarrow \left(\text{Robust\_Setting\_comp}_{expt}\right)_{i,0} \\ A \end{array} \right.$$

$$\text{Robust\_Setting} = \begin{pmatrix} -1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \qquad \text{Robust\_Setting\_comp}_2 = \begin{pmatrix} -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

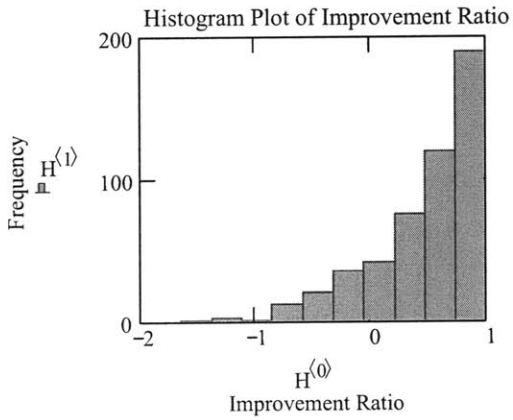
Finding the Variance at Predicted Robust Setting by Compound Noise

$$\text{Predicted\_Variance}_{expt} := \left| \begin{array}{l} \text{for } cfruns \in 0..(128-1) \\ \quad \left| \begin{array}{l} setting_{expt} \leftarrow \left| \begin{array}{l} \text{for } i \in 1..7 \\ \quad A_{(i-1),0} \leftarrow \left[\left(\text{Actual\_Robust\_Setting}_{expt}^{T} \right)^{\langle cfruns \rangle} \right]_{i,0} \\ A \end{array} \right. \\ Var \leftarrow \left[\left(\text{Actual\_Robust\_Setting}_{expt}^{T} \right)^{\langle cfruns \rangle} \right]_{0,0} \\ \text{break if } \left(setting_{expt} = \text{Robust\_Setting\_comp}_{expt} \right) \end{array} \right. \\ Var \end{array} \right.$$

Defining Improvement Ratio

$$\text{Improvement\_Ratio}_{expt} := \frac{\left(\text{Average\_Variance}_{expt} - \text{Predicted\_Variance}_{expt} \right)}{\left(\text{Average\_Variance}_{expt} - \text{Minimum\_Variance}_{expt} \right)}$$

$H := \text{histogram}(10, \text{Improvement\_Ratio})$



$\text{mean}(\text{Improvement\_Ratio}) = 0.479$

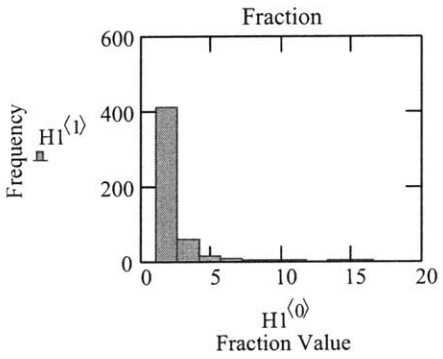
$\text{median}(\text{Improvement\_Ratio}) = 0.599$

Overall Results

Defining Ratio of Predicted Variance to Minimum Variance

$$Fraction1_{expt} := \frac{Predicted\_Variance_{expt}}{Minimum\_Variance_{expt}}$$

H1 := histogram(10, Fraction1)



mean(Fraction1) = 2.031

median(Fraction1) = 1.561

14.1

```
% function surrogation()

% We first assume the model parameters we want to use in Fitted WH Strong Hierarchy
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Noise Surrogation is done by finding 2 most important noises out of the
% total 5 noises for a given system based on their absolute scale

% Base Standard Deviation is taken as the average of standard deviations at
% all CF settings.

% We will find optimal control factor setting for surrogated model using
% Transmitted Variance Model. And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and Fitted WH
% model determines how active they are.

% Fitted WH for 200 models for Strong Hierarchy Fitted WH Model

% 09/29/2004 by Jagmeet Singh


clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g');  % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d e');      % defining 2(5) Full Factorial Array for NF's



modelpara=4;    % Defining which model parameters we would be using for Strong Hierarchy
model
% Basic WH(1);  Basic low w(2); Basic 2nd order(3); Fitted WH(4);   Fitted
% low order(5); Fitted 2nd order(6)
modelparameter=models(modelpara);      % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1);     s1=modelparameter(1,2);     s2=modelparameter(1,3);
w1=modelparameter(1,4);    w2=modelparameter(1,5);     p=modelparameter(1,6);
p11=modelparameter(1,7);   p01=modelparameter(1,8);    p00=modelparameter(1,9);% defining
parameters

ncf=7;  % # of CF's
nnf=5;  % # of NF's
```

```
counter_OCF_1 = 0;  % To increment when OCF from MC is same as from Noise Surrogation

MU = [0    0    0    0    0];   % Defines the means of the Noise Variables been used
sigma_uncorrelated = eye(5);    % Define the covariance matrix for the assumed model
% the on diagonal elements are ones and rest all zero. EYE function
% generates Identity Matrix


maxnoisevar = 200;    % Maximum number of Noise Factor settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;        % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h1 = waitbar(0,'Running Models');
for modelcounter=1:maxmodels      % To run a given number of models
   [bi,bij]=RWH_2ndorder(ncf,nnf,c,s1,w1,p,p11,p01,p00);    % Finding beta values for a given
model

   [main_noises, indices] = sort(abs(bi(1:5))); % To sort-out main 2 noises out of 5 based on
absolute scale
   % 4th and 5th element of indices will give the indices of main 2 noises




   % For 2(5) Full Factorial Noise Array
   clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
   index_resp_matrix = 1;  % Counter for response matrix
   for cfruns = 1:128
      for nfruns = 1:32
         x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
         x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
         sumij=0; sumijk=0;
         for i=1:(nnf+ncf)
            for j=i+1:(nnf+ncf)
               sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
            end
         end

         y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
         ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
         index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
   end
```

```
% Changing ResponseMatrix1 to [N1 N2 C1:C7 Y] where N1 and N2 are two
% main noises and C1:C7 are control Factors and Y is the response we
% got from above and naming it ResponseMatrix_NS

ResponseMatrix_NS = [ResponseMatrix1(:,indices(4)) ResponseMatrix1(:,indices(5))
ResponseMatrix1(:,6:13)];



clear NxC NxN CxCxN CxNxN;     % Clearing the History
% Fitting Response Model to 'y1' for 2 main noises

nxc=1;     % Counter for Control by Noise Interactions
for nf=1:2
   for cf=3:9
      NxC(:,nxc)=ResponseMatrix_NS(:,nf).*ResponseMatrix_NS(:,cf);
      nxc=nxc+1;
   end
end
nxn=1;     % Counter for Noise by Noise Interactions
for nf1=1:2
   for nf2=nf1+1:2
      NxN(:,nxn)=ResponseMatrix_NS(:,nf1).*ResponseMatrix_NS(:,nf2);
      nxn=nxn+1;
   end
end
cxnxn=1;          % Counter for Control X Noise X Noise Interaction
for nf1=1:2
   for nf2=nf1+1:2
      for cf = 3:9

CxNxN(:,cxnxn)=ResponseMatrix_NS(:,cf).*ResponseMatrix_NS(:,nf1).*ResponseMatrix_NS(:,nf
2);
         cxnxn = cxnxn + 1;
      end
   end
end
cxcxn=1;          % Counter for Control X Control X Noise Interaction
for nf=1:2
   for cf1=3:9
      for cf2=cf1+1:9

CxCxN(:,cxcxn)=ResponseMatrix_NS(:,cf1).*ResponseMatrix_NS(:,cf2).*ResponseMatrix_NS(:,n
f);
         cxcxn = cxcxn + 1;
      end
   end
end
```

```
% To find the fitted model for Transmitted Variance Model
inputs = [ones(4096,1) ResponseMatrix_NS(:,1:9) NxC NxN CxNxN CxCxN];
[b,bint,r,rint,stats]=regress(ResponseMatrix_NS(:,10),inputs);
% b0(1) bi's(2:10)  CxN(11:24)  NxN(25:25)  CxNxN(26:32) CxCxN(33:74) The way b's
% are defined

% To Determine the variance under each CF setting for Transmitted
% Response Model

X = ff2n(7)*2 - 1;  % Defining x's for Response Model


% Defining B's for the ease
Bi(1:9) = b(2:10);
Bij(2,9) = 0;

index=11;   % CxN
for i=1:2
   for j=3:9
      Bij(i,j) = b(index);
      index = index + 1;
   end
end

for i=1:2   % NxN
   for j=i+1:2
      Bij(i,j) = b(index);
      index = index + 1;
   end
end

Bijk(9,9,9) = 0;

for i=1:2   % CxNxN
   for j=i+1:2
      for k=3:9
         Bijk(i,j,k) = b(index);
         index = index + 1;
      end
   end
end

for i=1:2   % CxCxN
   for j=3:9
      for k=j+1:9
         Bijk(i,j,k) = b(index);
         index = index + 1;
      end
   end
end
```

```
% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:2     % First term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
      for j=3:9
         sum_b=sum_b+Bij(nf,j)*X(cf,j-2);
      end
      sum_c=0;
      for j=3:9
         for k=j+1:9
            sum_c=sum_c+Bijk(nf,j,k)*X(cf,j-2)*X(cf,k-2);
         end
      end
      sum1 = sum1 + (sum_a + sum_b +sum_c)^2;
   end

   for nf = 1:2     % Second term in Variance Model
      for j=nf+1:2
         sum_d=0;
         for k=3:9
            sum_d=sum_d+Bijk(nf,j,k)*X(cf,k-2);
         end
         sum2 = sum2 + (Bij(nf,j) + sum_d)^2;
      end
   end

   varianc(cf,1) = sum1 + sum2;
end

STDev_1 = varianc.^0.5;      % Stdev for each CF setting
std_base_1 = STDev_1(1,1);
op_std_1 = min(STDev_1);    % Finding least Stdev
PredMin_1 = sortrows([STDev_1 X],1);
OCF_NS = PredMin_1(1,2:8);




% Doing Monte Carlo for each setting of Control Factors
clear ResponseMatrix_MC;    % Response Matrix = [CF Y's_for_CFsetting]
[ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij, MU, sigma_uncorrelated);

STDev_MC = varianc.^0.5;      % Stdev for each CF setting
```

```
   %std_base_MC = STDev_MC(1,1);
   std_base_MC = mean(STDev_MC);  % Base Stdev is taken as mean of all STDev's

   op_std_MC = min(STDev_MC);     % Finding least Stdev
   PredMin_MC = sortrows([STDev_MC X],1);
   OCF_MC = PredMin_MC(1,2:8);

   if OCF_MC == OCF_NS
      counter_OCF_1 = counter_OCF_1 + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and 2(5-1)(V) Noise Array
   end




   cf_NS = 0;     % To find number of Control factors whose settings are predicted correctly

   for matching_cf = 1:7
      if OCF_NS(1,matching_cf) == OCF_MC(1,matching_cf)
         cf_NS = cf_NS + 1;
      end
   end

   matching_surrogate(modelcounter) = cf_NS;  % To store # of OCF Matched




   % Determining the Optimal Standard Deviation from Monte Carlo
   Opt_MC = std_for_cfsetting(ResponseMatrix_MC, OCF_MC);

   % Determining the Optimal Standard Deviation from Noise Surrogation
   Opt_NS = std_for_cfsetting(ResponseMatrix_MC, OCF_NS);

   std_base = std_base_MC;     % Base Stdev is taken as mean of all STDev's


   % Storing and Analysing Results
   std_fraction1(modelcounter) = (Opt_NS / Opt_MC);

   % Storing Improvement Ratios for Noise Surrogation
   std_fraction2(modelcounter) = ((std_base - Opt_NS)/(std_base - Opt_MC + 1e-10));

   Y(modelcounter) = (std_base - Opt_NS)/std_base;
   X1(modelcounter) = (std_base - Opt_MC)/std_base;

   waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d',modelcounter))
end
close(h1);  % Close waitbar
```

```
% saving workspace
save variables;

output;
```

14.2

```
function vector=models(modelpara)
% It defines the parameters that we would be using for Relaxed-Weak
% Heredity model
% Reference Chipman, Hamada and Wu paper 1997 and Li and Frey 2005 paper
% 03/04/2005 by Jagmeet Singh


Table1 = [10   1    1     1      1
      10   1    1     0.1    0.1
      10   1    0     1      1
      15   1/3  2/3   1      1
      15   1/3  2/3   0.1    0.1
      15   1/3  0     1      1];

Table2 = [0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0        0     0     0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0        0     0     0];

Table1= [Table1 Table2];      % for input to Main Model
vector=Table1(modelpara,:);
```

14.3

```
function [bi,bij]=RWH_2ndorder(ncf,nnf,c,s1,w1,p,p11,p01,p00)

% Developing 2nd order RWH Model (without ERROR)
% INPUTS: # of CF's,   # of NF's,     c,     s1,    w1,    p,
% p00,     p01,       p11

% OUTPUT: bi's and bij's
% Developed on 03/03/2004 by Jagmeet Singh

for i=1:(ncf+nnf)   % Defining t as mentioned in the writeup
   if i <= nnf
      t(i)=w1;
   else
      t(i)=1;
   end
end

delta=unifrnd(0,1,[1 nnf+ncf]);     % Defining delta
for i=1:nnf+ncf                  % Prob (delta_i = 1) = p
   if delta(1,i) <= p
      delta(1,i)=1;
   else
      delta(1,i)=0;
   end
end

deltaij(1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
   for j=i+1:(nnf+ncf)
      sum_deltas=delta(1,i)+delta(1,j);  % Finding the sum of delta-i + delta-j
      deltaij(i,j)=unifrnd(0,1);         % Defining delta-ij [0,1]

      if sum_deltas == 0               % Defining delta-ij when both main factors are inactive
         if deltaij(i,j) <= p00
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 1               % Defining delta-ij when one of the factors is active
         if deltaij(i,j) <= p01
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 2               % Defining delta-ij when both the factors are active
```

```
        if deltaij(i,j) <= p11
            deltaij(i,j)=1;
        else
            deltaij(i,j)=0;
        end
    end

  end
end

for i=1:nnf+ncf                    % Defining bi's for the CF's and NF's
  if delta(1,i) == 0
    bi(1,i)=t(i)*normrnd(0,1);
  else
    bi(1,i)=t(i)*normrnd(0,c);
  end
end

bij(1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
  for j=i+1:(nnf+ncf)
    if deltaij(i,j) == 0
      bij(i,j)=t(i)*t(j)*normrnd(0,s1);
    else
      bij(i,j)=t(i)*t(j)*normrnd(0,c*s1);
    end
  end
end
```

14.4

```
% Function to find the variance of the response once it is given the
% Control Factor setting and the Matrix with contains the response and
% cfsetting

% Inputs: Response Matrix and Required CF setting

% Output: Standard Deviation of Response for the given CF setting

% 03/24/2004 by Jagmeet Singh

function [std_dev] = std_for_cfsetting(ResponseMatrix, setting)

global cfsetting w2 ncf nnf maxnoisevar;

clear ysetting;

for i = 1:128
   if setting == ResponseMatrix(i,1:7)
      ysetting = ResponseMatrix(i,8:maxnoisevar+7);
      std_dev = std(ysetting);
   end
end
```

14.5

```
% Function Var_cf_setting takes inputs from RWH model, mean of Noises and
% the Covariance Matrix, which shows that Noise variables are independent.
% It then generates the Response for 200 Noise factors settings and finds
% the variance at each control factor setting for full model

% 03/16/2004 by Jagmeet Singh

function [ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij, MU, sigma)

global cfsetting w2 ncf nnf maxnoisevar;

X = ff2n(7)*2 - 1;  % Defining x's for Response Model
nfsetting = lhsnorm(MU, sigma, maxnoisevar);
for cfruns = 1:size(X,1)
   for nfruns = 1:maxnoisevar
      x(1,1:5)=nfsetting(nfruns,:);  % Defining Noise Factor Settings
      x(1,6:12)=X(cfruns,:);  % Defining CF Settings
      sumij=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end
      y_MC(cfruns,nfruns)=bi*x' + sumij + normrnd(0,w2);
   end
end

ResponseMatrix_MC = [X y_MC];  % Storing CF setting and Yassumed for that setting
varianc = (var(y_MC'))';  % Finding the Variance for each CF setting
```

14.6

```
% Function to plot the outputs of the response

% 03/04/2005 by Jagmeet Singh

function output()

load variables;

% Plotting and Analysing Results from the runs

% Pie Chart when OCF_MC is same as OCF_N1
figure; success=[(counter_OCF_1) (maxmodels-counter_OCF_1)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Same CF levels: ';'Diff CF levels: '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in prediction of OCF._M_C from OCF._n_o_i_s_e _s_u_r_r_o_g_a_t_i_o_n
for',num2str(maxmodels),' models']);
hgsave('pie1');


% Plotting Histograms
figure;
hist(std_fraction1);
title(['Histogram for Fraction of OPT.STD_s_u_r_r_o_g_a_t_i_o_n to OPT.STD_M_C for
',num2str(maxmodels),' models']);
colormap cool; iq = prctile(std_fraction1,[25 50 75]); tmax = max(hist(std_fraction1));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('OPT.STD.NSurrogation / OPT.STD.MC');
ylabel(' number of systems ');
hgsave('fig1');

figure;
hist(std_fraction2);
title(['Histogram for Fraction of (STD_b_a_s_e -
OPT.STD_s_u_r_r_o_g_a_t_i_o_n)/(STD_b_a_s_e - OPT.STD_M_C) for ',num2str(maxmodels),'
models']);
colormap cool; iq = prctile(std_fraction2,[25 50 75]); tmax = max(hist(std_fraction2));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Base-OPT.STD.NSurrogation)/(STD.Base-OPT.STD.MC)');
ylabel(' number of systems ');
hgsave('fig2');
```

```
figure;
plot(X1,Y,'*','color','r');
X2 = [ones(size(X1')) X1'];
a = X2\Y';
Y2 = a'*X2';
B = [X1' Y2'];
i0 = regress(Y',X1');
B = sortrows(B,1);
hold on;
line([B(1,1);B(maxmodels,1)], [ B(1,2);B(maxmodels,2)],'Color','g', 'LineWidth', 0.5);
line([0;B(maxmodels,1)],[0;i0*B(maxmodels,1)],'LineWidth',1.5);
title(['For 2^n^d Order Model : Plotting (STD_b_a_s_e -
STD_s_u_r_r_o_g_a_t_i_o_n)/STD_b_a_s_e vs (STD_b_a_s_e - STD_o_p_t)/STD_b_a_s_e
and slope = ', num2str(a(2,1))]);
xlabel('(STD_b_a_s_e - STD_o_p_t)/STD_b_a_s_e');
ylabel('(STD_b_a_s_e - STD_s_u_r_r_o_g_a_t_i_o_n)/STD_b_a_s_e');
hgsave('fig3');

prob_pos = 0;   % To find the probability that compounding will yield positive improvement
for index = 1:maxmodels
   if Y(1,index) >= 0.00
      prob_pos = prob_pos + 1;
   end
end

% Printing the results
sprintf(['mean(Number of OCF matched for Noise Surrogation) =
',num2str(mean(matching_surrogate))])
sprintf([' Probability that Noise Surrogation will Yield Positive Improvement =
',num2str(prob_pos/maxmodels)])
```

15.1

```
% function surrogation()

% We first assume the model parameters we want to use in Fitted WH Weak Hierarchy
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Noise Surrogation is done by finding 2 most important noises out of the
% total 5 noises for a given system based on their absolute scale

% Base Standard Deviation is taken as the average of standard deviations at
% all CF settings.

% We will find optimal control factor setting for surrogated model using
% Transmitted Variance Model. And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and Fitted WH
% model determines how active they are.

% RWH for 200 models for Weak Hierarchy Fitted WH Model

% 09/29/2004 by Jagmeet Singh


clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g');  % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d e');     % defining 2(5) Full Factorial Array for NF's



modelpara=4;    % Defining which model parameters we would be using for Weak Hierarchy
model
% Basic WH(1);  Basic low w(2); Basic 2nd order(3); Fitted WH(4);   Fitted
% low order(5); Fitted 2nd order(6)
modelparameter=models(modelpara);      % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1);     s1=modelparameter(1,2);     s2=modelparameter(1,3);
w1=modelparameter(1,4);    w2=modelparameter(1,5);     p=modelparameter(1,6);
p11=modelparameter(1,7);   p01=modelparameter(1,8);    p00=modelparameter(1,9);
p111=modelparameter(1,10); p011=modelparameter(1,11); p001=modelparameter(1,12);
p000=modelparameter(1,13);% defining parameters

ncf=7;  % # of CF's
```

```
nnf=5;  % # of NF's


counter_OCF_1 = 0;  % To increment when OCF from MC is same as from Noise Surrogation

MU = [0    0     0     0     0];    % Defines the means of the Noise Variables been used
sigma_uncorrelated = eye(5);     % Define the covariance matrix for the assumed model
% the on diagonal elements are ones and rest all zero. EYE function
% generates Identity Matrix



maxnoisevar = 200;     % Maximum number of Noise Factor settings
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;         % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h1 = waitbar(0,'Running Models');
for modelcounter=1:maxmodels       % To run a given number of models
   [bi,bij,bijk]=RWH_3rdorder(ncf,nnf,c,s1,s2,w1,p,p11,p01,p00,p111,p011,p001,p000);    %
Finding beta values for a given model

   [main_noises, indices] = sort(abs(bi(1:5))); % To sort-out main 2 noises out of 5 based on
absolute scale
   % 4th and 5th element of indices will give the indices of main 2 noises




   % For 2(5) Full Factorial Noise Array
   clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
   index_resp_matrix = 1;  % Counter for response matrix
   for cfruns = 1:128
      for nfruns = 1:32
         x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
         x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
         sumij=0; sumijk=0;
         for i=1:(nnf+ncf)
            for j=i+1:(nnf+ncf)
               sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
            end
         end
         for i=1:(nnf+ncf)
            for j=i+1:(nnf+ncf)
               for k=j+1:(nnf+ncf)
                  sumijk=sumijk+bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
               end
            end
         end
         y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
         ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
```

```
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end



    % Changing ResponseMatrix1 to [N1 N2 C1:C7 Y] where N1 and N2 are two
    % main noises and C1:C7 are control Factors and Y is the response we
    % got from above and naming it ResponseMatrix_NS

    ResponseMatrix_NS = [ResponseMatrix1(:,indices(4)) ResponseMatrix1(:,indices(5))
ResponseMatrix1(:,6:13)];




    clear NxC NxN CxCxN CxNxN;      % Clearing the History
    % Fitting Response Model to 'y1' for 2 main noises

    nxc=1;      % Counter for Control by Noise Interactions
    for nf=1:2
      for cf=3:9
        NxC(:,nxc)=ResponseMatrix_NS(:,nf).*ResponseMatrix_NS(:,cf);
        nxc=nxc+1;
      end
    end
    nxn=1;      % Counter for Noise by Noise Interactions
    for nf1=1:2
      for nf2=nf1+1:2
        NxN(:,nxn)=ResponseMatrix_NS(:,nf1).*ResponseMatrix_NS(:,nf2);
        nxn=nxn+1;
      end
    end
    cxnxn=1;          % Counter for Control X Noise X Noise Interaction
    for nf1=1:2
      for nf2=nf1+1:2
        for cf = 3:9

CxNxN(:,cxnxn)=ResponseMatrix_NS(:,cf).*ResponseMatrix_NS(:,nf1).*ResponseMatrix_NS(:,nf
2);
          cxnxn = cxnxn + 1;
        end
      end
    end
    cxcxn=1;          % Counter for Control X Control X Noise Interaction
    for nf=1:2
      for cf1=3:9
        for cf2=cf1+1:9
```

```
CxCxN(:,cxcxn)=ResponseMatrix_NS(:,cf1).*ResponseMatrix_NS(:,cf2).*ResponseMatrix_NS(:,n
f);
            cxcxn = cxcxn + 1;
        end
    end
end

% To find the fitted model for Transmitted Variance Model
inputs = [ones(4096,1) ResponseMatrix_NS(:,1:9) NxC NxN CxNxN CxCxN];
[b,bint,r,rint,stats]=regress(ResponseMatrix_NS(:,10),inputs);
% b0(1) bi's(2:10)  CxN(11:24)  NxN(25:25)  CxNxN(26:32) CxCxN(33:74) The way b's
% are defined

% To Determine the variance under each CF setting for Transmitted
% Response Model

X = ff2n(7)*2 - 1;  % Defining x's for Response Model


% Defining B's for the ease
Bi(1:9) = b(2:10);
Bij(2,9) = 0;

index=11;   % CxN
for i=1:2
    for j=3:9
        Bij(i,j) = b(index);
        index = index + 1;
    end
end

for i=1:2   % NxN
    for j=i+1:2
        Bij(i,j) = b(index);
        index = index + 1;
    end
end

Bijk(9,9,9) = 0;

for i=1:2   % CxNxN
    for j=i+1:2
        for k=3:9
            Bijk(i,j,k) = b(index);
            index = index + 1;
        end
    end
end

for i=1:2   % CxCxN
```

```
   for j=3:9
      for k=j+1:9
         Bijk(i,j,k) = b(index);
         index = index + 1;
      end
   end
end




% Fitting Transmitted Variance Model
for cf = 1:128
   sum1=0;sum2=0;
   for nf = 1:2      % First term in Variance Model
      sum_a=Bi(nf);
      sum_b=0;
      for j=3:9
         sum_b=sum_b+Bij(nf,j)*X(cf,j-2);
      end
      sum_c=0;
      for j=3:9
         for k=j+1:9
            sum_c=sum_c+Bijk(nf,j,k)*X(cf,j-2)*X(cf,k-2);
         end
      end
      sum1 = sum1 + (sum_a + sum_b +sum_c)^2;
   end

   for nf = 1:2      % Second term in Variance Model
      for j=nf+1:2
         sum_d=0;
         for k=3:9
            sum_d=sum_d+Bijk(nf,j,k)*X(cf,k-2);
         end
         sum2 = sum2 + (Bij(nf,j) + sum_d)^2;
      end
   end

   varianc(cf,1) = sum1 + sum2;
end

STDev_1 = varianc.^0.5;       % Stdev for each CF setting
std_base_1 = STDev_1(1,1);
op_std_1 = min(STDev_1);    % Finding least Stdev
PredMin_1 = sortrows([STDev_1 X],1);
OCF_NS = PredMin_1(1,2:8);
```

```
% Doing Monte Carlo for each setting of Control Factors
clear ResponseMatrix_MC;    % Response Matrix = [CF Y's_for_CFsetting]
[ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij,bijk, MU, sigma_uncorrelated);

STDev_MC = varianc.^0.5;        % Stdev for each CF setting

%std_base_MC = STDev_MC(1,1);
std_base_MC = mean(STDev_MC);   % Base Stdev is taken as mean of all STDev's

op_std_MC = min(STDev_MC);      % Finding least Stdev
PredMin_MC = sortrows([STDev_MC X],1);
OCF_MC = PredMin_MC(1,2:8);

if OCF_MC == OCF_NS
   counter_OCF_1 = counter_OCF_1 + 1;  % When same Optimal CF setting is predicted by
Monte Carlo and 2(5-1)(V) Noise Array
end




cf_NS = 0;      % To find number of Control factors whose settings are predicted correctly

for matching_cf = 1:7
   if OCF_NS(1,matching_cf) == OCF_MC(1,matching_cf)
      cf_NS = cf_NS + 1;
   end
end

matching_surrogate(modelcounter) = cf_NS;   % To store # of OCF Matched




% Determining the Optimal Standard Deviation from Monte Carlo
Opt_MC = std_for_cfsetting(ResponseMatrix_MC, OCF_MC);

% Determining the Optimal Standard Deviation from Noise Surrogation
Opt_NS = std_for_cfsetting(ResponseMatrix_MC, OCF_NS);

std_base = std_base_MC;     % Base Stdev is taken as mean of all STDev's


% Storing and Analysing Results
std_fraction1(modelcounter) = (Opt_NS / Opt_MC);

% Storing Improvement Ratios for Noise Surrogation
std_fraction2(modelcounter) = ((std_base - Opt_NS)/(std_base - Opt_MC + 1e-10));
```

```
   Y(modelcounter) = (std_base - Opt_NS)/std_base;
   X1(modelcounter) = (std_base - Opt_MC)/std_base;

   waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d',modelcounter))
end
close(h1);  % Close waitbar

% saving workspace
save variables;

output;
```

15.2

```
function vector=models(modelpara)
% It defines the parameters that we would be using for Relaxed-Weak
% Heredity model
% Reference Chipman, Hamada and Wu paper 1997 and Li and Frey 2005 paper
% 03/04/2005 by Jagmeet Singh


Table1 = [10   1    1     1      1
      10   1    1     0.1    0.1
      10   1    0     1      1
      15   1/3  2/3   1      1
      15   1/3  2/3   0.1    0.1
      15   1/3  0     1      1];

Table2 = [0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0        0     0     0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0        0     0     0];

Table1= [Table1 Table2];      % for input to Main Model
vector=Table1(modelpara,:);
```

15.3

```
function [bi,bij,bijk]=RWH_3rdorder(ncf,nnf,c,s1,s2,w1,p,p11,p01,p00,p111,p011,p001,p000)

% Developing 3rd order RWH Model (without ERROR) Including the Demand and
% Capacity noises for Phase 5 study.

% INPUTS: # of CF's,   # of NF's,    c,    s1,   s2,   w1,   p,
% p00,    p01,      p11,         p111,  p011,  p001   p000

% OUTPUT: bi's, bij's,and bijk's
% Developed on 03/24/2004 by Jagmeet Singh

% Defining the intensity of Noise wrt range of Control Factor setting (w1)
w1 = 1.0;

for i=1:(ncf+nnf)  % Defining t as mentioned in the writeup
   if i <= nnf
      t(i)=w1;
   else
      t(i)=1;
   end
end



delta=unifrnd(0,1,[1 nnf+ncf]);    % Defining delta
for i=1:nnf+ncf                    % Prob (delta_i = 1) = p
   if delta(1,i) <= p
      delta(1,i)=1;
   else
      delta(1,i)=0;
   end
end

deltaij(1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
   for j=i+1:(nnf+ncf)
      sum_deltas=delta(1,i)+delta(1,j);  % Finding the sum of delta-i + delta-j
      deltaij(i,j)=unifrnd(0,1);         % Defining delta-ij [0,1]

      if sum_deltas == 0                 % Defining delta-ij when both main factors are inactive
         if deltaij(i,j) <= p00
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 1                 % Defining delta-ij when one of the factors is active
```

```
        if deltaij(i,j) <= p01
          deltaij(i,j)=1;
        else
          deltaij(i,j)=0;
        end
      end

      if sum_deltas == 2              % Defining delta-ij when both the factors are active
        if deltaij(i,j) <= p11
          deltaij(i,j)=1;
        else
          deltaij(i,j)=0;
        end
      end

    end
end

% Defining delta-ijk
deltaijk(1:(nnf+ncf),1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
  for j=i+1:(nnf+ncf)
    for k=j+1:(nnf+ncf)
      sum_deltas=delta(1,i)+delta(1,j)+delta(1,k);   % Finding the sum of delta's
      deltaijk(i,j,k)=unifrnd(0,1);                  % Defining delta-ijk [0,1]

      if sum_deltas == 0          % Defining delta-ijk when all 3 main effects are inactive
        if deltaijk(i,j,k) <= p000
          deltaijk(i,j,k)=1;
        else
          deltaijk(i,j,k)=0;
        end
      end

      if sum_deltas == 1          % Defining delta-ijk when all 2 main effects are inactive
        if deltaijk(i,j,k) <= p001
          deltaijk(i,j,k)=1;
        else
          deltaijk(i,j,k)=0;
        end
      end

      if sum_deltas == 2          % Defining delta-ijk when all 2 main effects are active
        if deltaijk(i,j,k) <= p011
          deltaijk(i,j,k)=1;
        else
          deltaijk(i,j,k)=0;
        end
      end

      if sum_deltas == 3          % Defining delta-ijk when all 3 main effects are active
        if deltaijk(i,j,k) <= p111
```

```
            deltaijk(i,j,k)=1;
          else
            deltaijk(i,j,k)=0;
          end
        end

      end
    end
end


for i=1:nnf+ncf                    % Defining bi's for the CF's and NF's
    if delta(1,i) == 0
      bi(1,i)=t(i)*normrnd(0,1);
    else
      bi(1,i)=t(i)*normrnd(0,c);
    end
end

bij(1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
    for j=i+1:(nnf+ncf)
      if deltaij(i,j) == 0
        bij(i,j)=t(i)*t(j)*normrnd(0,s1);
      else
        bij(i,j)=t(i)*t(j)*normrnd(0,c*s1);
      end
    end
end

bijk(1:nnf+ncf,1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
    for j=i+1:nnf+ncf
      for k=j+1:nnf+ncf
        if deltaijk(i,j,k) == 0
          bijk(i,j,k)=t(i)*t(j)*t(k)*normrnd(0,s2);
        else
          bijk(i,j,k)=t(i)*t(j)*t(k)*normrnd(0,c*s2);
        end
      end
    end
end
```

15.4

```
% Function to find the variance of the response once it is given the
% Control Factor setting and the Matrix with contains the response and
% cfsetting

% Inputs: Response Matrix and Required CF setting

% Output: Standard Deviation of Response for the given CF setting

% 03/24/2004 by Jagmeet Singh

function [std_dev] = std_for_cfsetting(ResponseMatrix, setting)

global cfsetting w2 ncf nnf maxnoisevar;

clear ysetting;

for i = 1:128
   if setting == ResponseMatrix(i,1:7)
      ysetting = ResponseMatrix(i,8:maxnoisevar+7);
      std_dev = std(ysetting);
   end
end
```

15.5

```
% Function Var_cf_setting takes inputs from RWH model, mean of Noises and
% the Covariance Matrix, which shows that Noise variables are independent.
% It then generates the Response for 200 Noise factors settings and finds
% the variance at each control factor setting for full model

% 08/10/2004 by Jagmeet Singh

function [ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij, bijk, MU, sigma)

global cfsetting w2 ncf nnf maxnoisevar;

X = ff2n(7)*2 - 1;  % Defining x's for Response Model
nfsetting = lhsnorm(MU, sigma, maxnoisevar);
for cfruns = 1:size(X,1)
   for nfruns = 1:maxnoisevar
      x(1,1:5)=nfsetting(nfruns,:);   % Defining Noise Factor Settings
      x(1,6:12)=X(cfruns,:);  % Defining CF Settings
      sumij=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end
      sumijk=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            for k=j+1:(nnf+ncf)
               sumijk=sumijk + bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
            end
         end
      end
      y_MC(cfruns,nfruns)=bi*x' + sumij + sumijk + normrnd(0,w2);
   end
end

ResponseMatrix_MC = [X y_MC];   % Storing CF setting and Yassumed for that setting
varianc = (var(y_MC'))';   % Finding the Variance for each CF setting
```

15.6

```
% Function to plot the outputs of the response

% 09/29/2004 by Jagmeet Singh

function output()

load variables;

% Plotting and Analysing Results from the runs

% Pie Chart when OCF_MC is same as OCF_N1
figure; success=[(counter_OCF_1) (maxmodels-counter_OCF_1)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Same CF levels: ';'Diff CF levels: '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in prediction of OCF._M_C from OCF._n_o_i_s_e _s_u_r_r_o_g_a_t_i_o_n
for',num2str(maxmodels),' models']);
hgsave('pie1');


% Plotting Histograms
figure;
hist(std_fraction1);
title(['Histogram for Fraction of OPT.STD_s_u_r_r_o_g_a_t_i_o_n to OPT.STD_M_C for
',num2str(maxmodels),' models']);
colormap cool; iq = prctile(std_fraction1,[25 50 75]); tmax = max(hist(std_fraction1));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('OPT.STD.NSurrogation / OPT.STD.MC');
ylabel(' number of systems ');
hgsave('fig1');

figure;
hist(std_fraction2);
title(['Histogram for Fraction of (STD_b_a_s_e -
OPT.STD_s_u_r_r_o_g_a_t_i_o_n)/(STD_b_a_s_e - OPT.STD_M_C) for ',num2str(maxmodels),'
models']);
colormap cool; iq = prctile(std_fraction2,[25 50 75]); tmax = max(hist(std_fraction2));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Base-OPT.STD.NSurrogation)/(STD.Base-OPT.STD.MC)');
ylabel(' number of systems ');
hgsave('fig2');
```

```
figure;
plot(X1,Y,'*','color','r');
X2 = [ones(size(X1')) X1'];
a = X2\Y';
Y2 = a'*X2';
B = [X1' Y2'];
i0 = regress(Y',X1');
B = sortrows(B,1);
hold on;
line([B(1,1);B(maxmodels,1)], [ B(1,2);B(maxmodels,2)],'Color','g', 'LineWidth', 0.5);
line([0;B(maxmodels,1)],[0;i0*B(maxmodels,1)],'LineWidth',1.5);
title(['For 3^r^d Order Model : Plotting (STD_b_a_s_e -
STD_s_u_r_r_o_g_a_t_i_o_n)/STD_b_a_s_e vs (STD_b_a_s_e - STD_o_p_t)/STD_b_a_s_e
and slope = ', num2str(a(2,1))]);
xlabel('(STD_b_a_s_e - STD_o_p_t)/STD_b_a_s_e');
ylabel('(STD_b_a_s_e - STD_s_u_r_r_o_g_a_t_i_o_n)/STD_b_a_s_e');
hgsave('fig3');

prob_pos = 0;   % To find the probability that compounding will yield positive improvement
for index = 1:maxmodels
   if Y(1,index) >= 0.00
      prob_pos = prob_pos + 1;
   end
end

% Printing the results
sprintf(['mean(Number of OCF matched for Noise Surrogation) =
',num2str(mean(matching_surrogate))])
sprintf([' Probability that Noise Surrogation will Yield Positive Improvement =
',num2str(prob_pos/maxmodels)])
```

16.1

```
% We first assume the model parameters we want to use in RWH 2nd order
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model.

% Noise factors are taken as independent.

% Noise Surrogation (TTBF) is done by finding 2 most important noises out of the
% total 5 noises for a given system based on their absolute scale

% We will find optimal control factor setting for TTBF noise strategy.
% And compare that with optimal control factor
% setting got from using Monte Carlo to generate noise factor settings. We
% will run this for 200 models. Each model has 7 CF's and 5 NF's and RWH
% model determines how active they are.

% Find Improvement Ratio for each value of p11, p01, p00 from 0.01 to 1.00.
% The p (prob. of active main effects) = 0.95. Since for most of the Strong
% Hierarchy case studies main effects were active with high probability.

% RWH for 200 models for 2nd order RWH Model

% 02/11/2006 by Jagmeet Singh

clear; clc;
global cfsetting w2 ncf nnf maxnoisevar;

[cfsetting,conf_cfsetting]=fracfact('a b c d e f g'); % defining 2(7) Full Factorial Array for CF's
[nfsetting,conf_nfsetting]=fracfact('a b c d e');      % defining 2(5) Full Factorial Array for NF's
[reference,conf_reference]=fracfact('a b');    % defining 2(2) Full Factorial for TTBF strategy



modelpara=6;   % Defining which model parameters we would be using for 2nd order model
% Basic WH(1); Basic low w(2); Basic 2nd order(3); Fitted WH(4);   Fitted
% low order(5); Fitted 2nd order(6)

modelparameter=models(modelpara);      % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1); s1=modelparameter(1,2); s2=modelparameter(1,3);
w1=modelparameter(1,4); w2=modelparameter(1,5);   p = 0.95;% defining parameters and
Changing 'p'

ncf=7;  % # of CF's
nnf=5;  % # of NF's
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;        % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

counter_p11 = 1;

for p11 = 0.01: 0.01: 1.00
X2(counter_p11) = p11;      % Store p11 values for final plot
p01 = p11;     % defining new probability parameters
p00 = p11;

  h1 = waitbar(0,'Running Models');
  for modelcounter=1:maxmodels       % To run a given number of models
    [bi,bij]=RWH_2ndorder(ncf,nnf,c,s1,w1,p,p11,p01,p00);    % Finding beta values for a given
model

    [main_noises, indices] = sort(abs(bi(1:5))); % To sort-out main 2 noises out of 5 based on
absolute scale
    % 4th and 5th element of indices will give the indices of main 2 noises

    % Filling nfsetting_ttbf according to discussed strategy
    for nfruns = 1:nnf
      for nfrows = 1:4
        dummy = randperm(2)*2 - 3;
        nfsetting_TTBF(nfrows, nfruns) = dummy(1,1);
      end
    end
    nfsetting_TTBF(:, indices(4))= reference(:,1);
    nfsetting_TTBF(:, indices(5))= reference(:,2);

    nfsetting_TTBF




    % For 2(5) Full Factorial Noise Array
    clear ResponseMatrix1;  % Response Matrix = [NF CF Y]
    index_resp_matrix = 1;  % Counter for response matrix
    for cfruns = 1:128
      for nfruns = 1:32
        x(1,1:5)=nfsetting(nfruns,:); % Defining Noise Factor Settings
        x(1,6:12)=cfsetting(cfruns,:);   % Defining CF Setting
        sumij=0; sumijk=0;
        for i=1:(nnf+ncf)
          for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
          end
        end
        y1(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
        ResponseMatrix1(index_resp_matrix,:)=[nfsetting(nfruns,:) cfsetting(cfruns,:)
y1(cfruns,nfruns)];
```

```
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end


    varianc = var(y1')';
    STDev_1 = varianc.^0.5;      % Stdev for each CF setting
    std_base_1 = STDev_1(1,1);
    op_std_1 = min(STDev_1);    % Finding least Stdev
    PredMin_1 = sortrows([STDev_1 cfsetting],1);
    OCF_N1 = PredMin_1(1,2:8);









    % For TTBF Noise Strategy at 4 levels
    clear ResponseMatrix_c;  % Response Matrix = [NF CF Y]
    index_resp_matrix = 1;  % Counter for response matrix
    for cfruns = 1:128
      for nfruns = 1:4
        x(1,1:5)=nfsetting_TTBF(nfruns,:); % Defining Noise Factor Settings
        x(1,6:12)=cfsetting(cfruns,:);    % Defining CF Setting
        sumij=0; sumijk=0;
        for i=1:(nnf+ncf)
          for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
          end
        end
        y_c(cfruns,nfruns) = bi*x' + sumij + sumijk + normrnd(0,w2);
        ResponseMatrix_c(index_resp_matrix,:)=[reference(nfruns,:) cfsetting(cfruns,:)
y_c(cfruns,nfruns)];
        index_resp_matrix = index_resp_matrix + 1;  % For Storing Response Matrix
      end
    end


    varianc = var(y_c')';
    STDev_c = varianc.^0.5;     % Stdev for each CF setting
    std_base_c = STDev_c(1,1);
    op_std_c = min(STDev_c);    % Finding Least STDev_1
    PredMin_c = sortrows([STDev_c cfsetting],1);
    OCF_c = PredMin_c(1,2:8);

    % Finding Optimal Standard Deviation from Compound Noise
    for cfruns = 1:128
```

```
        if OCF_c == PredMin_1(cfruns, 2:8);
           Opt_c = PredMin_1(cfruns,1);
        end
      end

      % Determining the Optimal Standard Deviation from Noise Strategy 1
      Opt_1 = op_std_1;

      std_base = mean(STDev_1);    % Base Stdev is taken as mean of all STDev's


      % Storing Improvement Ratios for Compound Noise
      std_fraction4(modelcounter) = ((std_base - Opt_c)/(std_base - Opt_1 + 1e-10));


      waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d for p11, p01, p00 =
%.2f',modelcounter, p11))
   end
   close(h1);  % Close waitbar

   improvement_ratio_mean(counter_p11) = mean(std_fraction4);      % Finding Improvement
Ratio for given p11
   improvement_ratio_median(counter_p11) = median(std_fraction4);      % Finding Improvement
Ratio for given p11
   counter_p11 = counter_p11 + 1;     % Increasing the Counter

end


% saving workspace
save variables;

clear; clc;
load variables;    % to remove previous data and upload the current data

% Plotting Improvement Ratio Mean vs P11, P01, P00

t2 = polyfit(X2, improvement_ratio_mean, 3);       % Fitting a 3rd order polynomial
y2 = polyval(t2,X2);
hold on;
plot(X2, improvement_ratio_mean, '.');
```

```
plot(X2, y2,'k','LineWidth',1, 'Marker', '+', 'MarkerEdgeColor','k',...
            'MarkerFaceColor','k',...
            'MarkerSize', 2);
xlabel('p_1_1, p_0_1, p_0_0', 'FontSize', 11);
ylabel('Mean Improvement Ratio',  'FontSize', 11);
title('Mean Improvement Ratio vs Density of Effects for RWH Model', 'FontSize',12);
ylim([0 1]);
hgsave('mean_improvement_ratio');
hold off;
figure;


% Plotting Improvement Ratio Median vs P11, P01, P00

t3 = polyfit(X2, improvement_ratio_median, 3);      % Fitting a 3rd order polynomial
y3 = polyval(t3,X2);
hold on;
plot(X2, improvement_ratio_median, '.');
plot(X2, y3,'k','LineWidth',1, 'Marker', '+', 'MarkerEdgeColor','k',...
            'MarkerFaceColor','k',...
            'MarkerSize', 2);
xlabel('p_1_1, p_0_1, p_0_0', 'FontSize', 11);
ylabel('Median Improvement Ratio',  'FontSize', 11);
title('Median Improvement Ratio vs Density of Effects for RWH Model', 'FontSize',12);
ylim([0 1]);
hgsave('median_improvement_ratio');
hold off;
```

16.2

```
function vector=models(modelpara)
% It defines the parameters that we would be using for Relaxed-Weak
% Heredity model
% Reference Chipman, Hamada and Wu paper 1997 and Prof. Frey's paper
% 03/04/2004 by Jagmeet Singh


Table1 = [10   1    1     1      1
      10   1    1     0.1    0.1
      10   1    0     1      1
      15   1/3  2/3   1      1
      15   1/3  2/3   0.1    0.1
      15   1/3  0     1      1];

Table2 = [0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0        0     0     0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0        0     0     0];

Table1= [Table1 Table2];      % for input to Main Model
vector=Table1(modelpara,:);
```

16.3

```
function [bi,bij]=RWH_2ndorder(ncf,nnf,c,s1,w1,p,p11,p01,p00)

% Developing 2nd order RWH Model (without ERROR)
% INPUTS: # of CF's,   # of NF's,    c,    s1,    w1,    p,
% p00,    p01,       p11

% OUTPUT: bi's and bij's
% Developed on 03/03/2004 by Jagmeet Singh

for i=1:(ncf+nnf)  % Defining t as mentioned in the writeup
   if i <= nnf
      t(i)=w1;
   else
      t(i)=1;
   end
end

delta=unifrnd(0,1,[1 nnf+ncf]);    % Defining delta
for i=1:nnf+ncf                    % Prob (delta_i = 1) = p
   if delta(1,i) <= p
      delta(1,i)=1;
   else
      delta(1,i)=0;
   end
end

deltaij(1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
   for j=i+1:(nnf+ncf)
      sum_deltas=delta(1,i)+delta(1,j);  % Finding the sum of delta-i + delta-j
      deltaij(i,j)=unifrnd(0,1);         % Defining delta-ij [0,1]

      if sum_deltas == 0                 % Defining delta-ij when both main factors are inactive
         if deltaij(i,j) <= p00
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 1                 % Defining delta-ij when one of the factors is active
         if deltaij(i,j) <= p01
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 2                 % Defining delta-ij when both the factors are active
```

```
        if deltaij(i,j) <= p11
           deltaij(i,j)=1;
        else
           deltaij(i,j)=0;
        end
     end

  end
end

for i=1:nnf+ncf                    % Defining bi's for the CF's and NF's
  if delta(1,i) == 0
     bi(1,i)=t(i)*normrnd(0,1);
  else
     bi(1,i)=t(i)*normrnd(0,c);
  end
end

bij(1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
  for j=i+1:(nnf+ncf)
     if deltaij(i,j) == 0
        bij(i,j)=t(i)*t(j)*normrnd(0,s1);
     else
        bij(i,j)=t(i)*t(j)*normrnd(0,c*s1);
     end
  end
end
```

16.4

```
% Function to find the variance of the response once it is given the
% Control Factor setting and the Matrix with contains the response and
% cfsetting

% Inputs: Response Matrix and Required CF setting

% Output: Standard Deviation of Response for the given CF setting

% 03/24/2004 by Jagmeet Singh

function [std_dev] = std_for_cfsetting(ResponseMatrix, setting)

global cfsetting w2 ncf nnf maxnoisevar;

clear ysetting;

for i = 1:128
   if setting == ResponseMatrix(i,1:7)
      ysetting = ResponseMatrix(i,8:maxnoisevar+7);
      std_dev = std(ysetting);
   end
end
```

16.5

```
% Function Var_cf_setting takes inputs from RWH model, mean of Noises and
% the Covariance Matrix, which shows that Noise variables are independent.
% It then generates the Response for 1000 Noise factors settings and finds
% the variance at each control factor setting for full model

% 03/16/2004 by Jagmeet Singh

function [ResponseMatrix_MC, varianc] = Var_cf_setting(bi, bij, MU, sigma)

global cfsetting w2 ncf nnf maxnoisevar;

X = ff2n(7)*2 - 1;  % Defining x's for Response Model
nfsetting = lhsnorm(MU, sigma, maxnoisevar);
for cfruns = 1:size(X,1)
   for nfruns = 1:maxnoisevar
      x(1,1:5)=nfsetting(nfruns,:);   % Defining Noise Factor Settings
      x(1,6:12)=X(cfruns,:);  % Defining CF Settings
      sumij=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end
      y_MC(cfruns,nfruns)=bi*x' + sumij + normrnd(0,w2);
   end
end

ResponseMatrix_MC = [X y_MC];   % Storing CF setting and Yassumed for that setting
varianc = (var(y_MC'))';   % Finding the Variance for each CF setting
```

17.1

```
% function effectcorrelation(): To check the impact of correlation and
% variance of noise factors

% We first assume the model parameters we want to use in RWH 3rd order
% model. We call that set of parameters from modelparameters.m. Then we
% find betas for a given model. We run 2 kinds of analysis on the beta
% values.

% One is the Assumed model in which noise factors are independent and the
% other one is Actual model in which noise factors are correlated. We find
% optimal CF setting in both cases. And the respective standard deviations
% at the optimal CF setting in both the cases. We runs this for 200
% models. Each model has 7 CF's and 5 NF's and RWH model determines how
% active they are. We will use MONTE CARLO method to find variance for each
% control factor setting

% BASIC WH for 200 models for 3rd order RWH Model

% 04/21/2005 by Jagmeet Singh

clear;clc;
global cfsetting w2 ncf nnf maxnoisevar;

cfsetting = ff2n(7)*2 - 1;      % Defining Full Factorial Design for the 7 Control Factors

modelpara=1;    % Defining which model parameters we would be using for 2nd order model
% Basic WH(1);  Basic low w(2); Basic 2nd order(3); Fitted WH(4);   Fitted
% low order(5); Fitted 2nd order(6)
modelparameter=models(modelpara);       % To get the values of c, s1, p's etc for the given
model
c=modelparameter(1,1);      s1=modelparameter(1,2);     s2=modelparameter(1,3);
w1=modelparameter(1,4);     w2=10.0                 ;     p=modelparameter(1,6);
p11=modelparameter(1,7);    p01=modelparameter(1,8);    p00=modelparameter(1,9);
p111=modelparameter(1,10);  p011=modelparameter(1,11);  p001=modelparameter(1,12);
p000=modelparameter(1,13);% defining parameters

ncf=7;  % # of CF's
nnf=5;  % # of NF's
counter_per_dev_op = 0;    % To increment when (std_actual - op_std_actual)/op_std_actual <
5%
counter_per_dev_base = 0;   % To increment when (std_actual - op_std_actual)/std_base_actual
< 5%
counter_OCF = 0;        % To increment when OCF_actual == OCF_assumed for a given model
run
MU = [0     0       0       0       0];     % Defines the means of the Noise Variables been used
sigma_assumed = eye(5);      % Define the covariance matrix for the assumed model
% the on diagonal elements are ones and rest all zero. EYE function
% generates Identity Matrix
maxnoisevar = 200;      % Maximum number of Noise Factor settings
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
maxmodels = 200;        % The number of models to be tested
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h1 = waitbar(0,'Running Models');
for modelcounter=1:maxmodels       % To run a given number of models
   modelcounter
   [bi,bij,bijk]=RWH_3rdorder(ncf,nnf,c,s1,s2,w1,p,p11,p01,p00,p111,p011,p001,p000);    %
Finding beta values for a given model

   % For Assumed Model with all independent noise factors
   clear ResponseMatrixAssumed;   % Response Matrix = [CF Yassumed's_for_CFsetting]
   [ResponseMatrixAssumed, varianc] = Assumedmodel(bi, bij, bijk, MU, sigma_assumed);

   STDev_assumed = varianc.^0.5;      % Stdev for each CF setting
   std_base_assumed = STDev_assumed(1,1);
   op_std_assumed = min(STDev_assumed);    % Finding least Stdev
   PredMin_assumed = sortrows([STDev_assumed cfsetting],1);
   OCF_assumed = PredMin_assumed(1,2:8);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % For Actual Model with correlated noise factors
   clear ResponseMatrixActual;    % Response Matrix = [CF Yactual's_for_CFsetting]

   % To generate Sigma_actual Matrix (Covariance Matrix) which has Correlation among
   % the noise variables and Has the same on diagonal elements as
   % sigma_assumed. And Sigma_Actual is Positive Semi Definite and
   % Symmetric.
   r = normrnd(0,0.01,[10 1]); % To generate random correlation coefficients
   sigma_actual = eye(5);   % Inital Matrix
   dummy_t=1;
   for i=1:5
      for j=i+1:5
         sigma_actual(i,j) = r(dummy_t,1);
         sigma_actual(j,i) = r(dummy_t,1);
         dummy_t = dummy_t+1;
      end
   end
   sigma_actual = corrcoef(sigma_actual);  % Gives PSD Covariance Symmetric Matrix
   for i=1:5
      for j=i+1:5
         sigma_actual(i,j) = sigma_actual(i,j)/25;
         sigma_actual(j,i) = sigma_actual(j,i)/25;
      end
   end
   % To reduce the correlation among noise factors


   [ResponseMatrixActual, varianc_actual] = Actualmodel(bi, bij, bijk, MU, sigma_actual);
```

```
STDev_actual = varianc_actual.^0.5;       % Stdev for each CF setting
std_base_actual = STDev_actual(1,1);
op_std_actual = min(STDev_actual);     % Finding least Stdev
PredMin_actual = sortrows([STDev_actual cfsetting],1);
OCF_actual = PredMin_actual(1,2:8);

        % Determining the Std_actual at OCF_assumed
         Std_actual = std_for_cfsetting(ResponseMatrixActual, OCF_assumed);

        %  We might select control factor settings randomly. So any
        %  setting of CF can be chosen to act as a base. Here we
        %  will find Standard deviation of a cf setting which is
        %  chosen at random and we will take it as one of the base
        %  standard deviation and we will work on the results
        base_cfsetting = unifrnd(-1,1,[1 7]);
        for index_cf=1:7
           if base_cfsetting(1,index_cf) <= 0.00
              base_cfsetting(1,index_cf)=-1;
           else
              base_cfsetting(1,index_cf)=1;
           end
        end
        std_base_actual_1 = std_for_cfsetting(ResponseMatrixActual, base_cfsetting);



% Storing and Analysing Results
per_dev_op(modelcounter) = ((Std_actual - op_std_actual)/op_std_actual)*100.;
if per_dev_op(modelcounter) < 5.000
   counter_per_dev_op = counter_per_dev_op + 1;  % When the difference is small
end
per_dev_base(modelcounter) = ((Std_actual - op_std_actual)/std_base_actual)*100.;
if per_dev_base(modelcounter) < 5.000
   counter_per_dev_base = counter_per_dev_base + 1;  % When the difference is small
end

if OCF_actual == OCF_assumed
   counter_OCF = counter_OCF + 1;        % When same CF setting is predicted in both cases
end

% Storing Optimal Standard Deviation and STD.Actual
OP_STD_ACTUAL(modelcounter) = op_std_actual;
STD_ACTUAL(modelcounter) = Std_actual;

% Storing % Improvement in Assumed Model is OCF_assumed are selected
improv_assumed(modelcounter) = ((std_base_assumed -
op_std_assumed)/std_base_assumed)*100.;
% Storing % Improvement in Actual Model if OCF_assumed are selected
improv_actual1(modelcounter) = ((std_base_actual - Std_actual)/std_base_actual)*100.;
% Storing % Improvement in Actual Model if OCF_actual are selected
```

```
   improv_actual2(modelcounter) = ((std_base_actual - op_std_actual)/std_base_actual)*100.;
   % Storing % Improvement in Actual Model if OCF_assumed are selected and
   % STD.Actual.1
   improv_actual3(modelcounter) = ((std_base_actual_1 - Std_actual)/std_base_actual_1)*100.;
   % Storing % Improvement in Actual Model if OCF_actual are selected and
   % STD.Actual.1
   improv_actual4(modelcounter) = ((std_base_actual_1 -
op_std_actual)/std_base_actual_1)*100.;


   % Storing Fractions of std_dev ratios for actual model
   std_fraction1(modelcounter) = (Std_actual/std_base_actual);
   % Storing Fractions of STD_Actual to Op_Std_Actual ratios of actual
   % model
   std_fraction2(modelcounter) = (Std_actual/op_std_actual);
   % Some Other important fractions
   std_fraction3(modelcounter) = ((Std_actual - std_base_actual_1)/(op_std_actual -
std_base_actual_1 + 1e-10));
            % To avoid division by zero

   std_fraction4(modelcounter) = ((Std_actual - std_base_actual)/(op_std_actual -
std_base_actual + 1e-10));
            % To avoid division by zero

   % Storing Fractions of std_dev ratios for actual model
   std_fraction5(modelcounter) = (Std_actual/std_base_actual_1);

   waitbar(modelcounter/maxmodels,h1,sprintf('Running Model #%d',modelcounter))
end
close(h1);  % Close waitbar

% saving workspace
save variables;

output;
```

17.2

```
function vector=models(modelpara)
% It defines the parameters that we would be using for Relaxed-Weak
% Heredity model
% Reference Chipman, Hamada and Wu paper 1997 and Prof. Frey's paper
% 03/04/2004 by Jagmeet Singh


Table1 = [10   1    1     1      1
      10   1    1     0.1    0.1
      10   1    0     1      1
      15   1/3  2/3   1      1
      15   1/3  2/3   0.1    0.1
      15   1/3  0     1      1];

Table2 = [0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0.25     0.1   0     0
      0.25    0.25     0.1   0     0        0     0     0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0.17     0.08  0.02  0
      0.43    0.31     0.04  0     0        0     0     0];

Table1= [Table1 Table2];      % for input to Main Model
vector=Table1(modelpara,:);
```

17.3

```
function [bi,bij,bijk]=RWH_3rdorder(ncf,nnf,c,s1,s2,w1,p,p11,p01,p00,p111,p011,p001,p000)

% Developing 3rd order RWH Model (without ERROR) Including the Demand and
% Capacity noises for Phase 5 study.

% INPUTS: # of CF's,   # of NF's,    c,    s1,    s2,    w1,    p,
% p00,    p01,       p11,          p111,  p011,  p001   p000

% OUTPUT: bi's, bij's,and bijk's
% Developed on 03/24/2004 by Jagmeet Singh

% Defining the intensity of Noise wrt range of Control Factor setting (w1)
w1 = 1.0;

for i=1:(ncf+nnf)  % Defining t as mentioned in the writeup
   if i <= nnf
      t(i)=w1;
   else
      t(i)=1;
   end
end



delta=unifrnd(0,1,[1 nnf+ncf]);    % Defining delta
for i=1:nnf+ncf                    % Prob (delta_i = 1) = p
   if delta(1,i) <= p
      delta(1,i)=1;
   else
      delta(1,i)=0;
   end
end

deltaij(1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
   for j=i+1:(nnf+ncf)
      sum_deltas=delta(1,i)+delta(1,j);  % Finding the sum of delta-i + delta-j
      deltaij(i,j)=unifrnd(0,1);         % Defining delta-ij [0,1]

      if sum_deltas == 0                 % Defining delta-ij when both main factors are inactive
         if deltaij(i,j) <= p00
            deltaij(i,j)=1;
         else
            deltaij(i,j)=0;
         end
      end

      if sum_deltas == 1                 % Defining delta-ij when one of the factors is active
```

```
        if deltaij(i,j) <= p01
           deltaij(i,j)=1;
        else
           deltaij(i,j)=0;
        end
     end

     if sum_deltas == 2              % Defining delta-ij when both the factors are active
        if deltaij(i,j) <= p11
           deltaij(i,j)=1;
        else
           deltaij(i,j)=0;
        end
     end

  end
end

% Defining delta-ijk
deltaijk(1:(nnf+ncf),1:(nnf+ncf),1:(nnf+ncf))=0;
for i=1:(nnf+ncf)
  for j=i+1:(nnf+ncf)
     for k=j+1:(nnf+ncf)
        sum_deltas=delta(1,i)+delta(1,j)+delta(1,k);   % Finding the sum of delta's
        deltaijk(i,j,k)=unifrnd(0,1);                  % Defining delta-ijk [0,1]

        if sum_deltas == 0           % Defining delta-ijk when all 3 main effects are inactive
           if deltaijk(i,j,k) <= p000
              deltaijk(i,j,k)=1;
           else
              deltaijk(i,j,k)=0;
           end
        end

        if sum_deltas == 1           % Defining delta-ijk when all 2 main effects are inactive
           if deltaijk(i,j,k) <= p001
              deltaijk(i,j,k)=1;
           else
              deltaijk(i,j,k)=0;
           end
        end

        if sum_deltas == 2           % Defining delta-ijk when all 2 main effects are active
           if deltaijk(i,j,k) <= p011
              deltaijk(i,j,k)=1;
           else
              deltaijk(i,j,k)=0;
           end
        end

        if sum_deltas == 3           % Defining delta-ijk when all 3 main effects are active
           if deltaijk(i,j,k) <= p111
```

```
            deltaijk(i,j,k)=1;
         else
            deltaijk(i,j,k)=0;
         end
      end

   end
end


for i=1:nnf+ncf                    % Defining bi's for the CF's and NF's
   if delta(1,i) == 0
      bi(1,i)=t(i)*normrnd(0,1);
   else
      bi(1,i)=t(i)*normrnd(0,c);
   end
end

bij(1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
   for j=i+1:(nnf+ncf)
      if deltaij(i,j) == 0
         bij(i,j)=t(i)*t(j)*normrnd(0,s1);
      else
         bij(i,j)=t(i)*t(j)*normrnd(0,c*s1);
      end
   end
end

bijk(1:nnf+ncf,1:nnf+ncf,1:nnf+ncf)=0;
for i=1:nnf+ncf
   for j=i+1:nnf+ncf
      for k=j+1:nnf+ncf
         if deltaijk(i,j,k) == 0
            bijk(i,j,k)=t(i)*t(j)*t(k)*normrnd(0,s2);
         else
            bijk(i,j,k)=t(i)*t(j)*t(k)*normrnd(0,c*s2);
         end
      end
   end
end
```

17.4

```
% Function to find the variance of the response once it is given the
% Control Factor setting and the Matrix with contains the response and
% cfsetting

% Inputs: Response Matrix and Required CF setting

% Output: Standard Deviation of Response for the given CF setting

% 03/24/2004 by Jagmeet Singh

function [std_dev] = std_for_cfsetting(ResponseMatrix, setting)

global cfsetting w2 ncf nnf maxnoisevar;

clear ysetting;

for i = 1:128
   if setting == ResponseMatrix(i,1:7)
      ysetting = ResponseMatrix(i,8:maxnoisevar+7);
      std_dev = std(ysetting);
   end
end
```

17.5

```
% Function Actual Model takes inputs from RWH model, mean of Noises and
% the Covariance Matrix, which shows that Noise variables are correlated.
% It then generates the Response for 200 Noise factors settings and finds
% the variance at each control factor setting for full model

% 03/24/2004 by Jagmeet Singh

function [ResponseMatrixActual, varianc] = Actualmodel(bi, bij, bijk, MU, sigma)

global cfsetting w2 ncf nnf maxnoisevar;

nfsetting = lhsnorm(MU, sigma, maxnoisevar);
for cfruns = 1:128
   for nfruns = 1:maxnoisevar
      x(1,1:5)=nfsetting(nfruns,:);   % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);  % Defining CF Settings

      sumij=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end

      sumijk=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            for k=j+1:(nnf+ncf)
               sumijk=sumijk + bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
            end
         end
      end

      yactual(cfruns,nfruns)=bi*x' + sumij + sumijk + normrnd(0,w2);
   end
end

ResponseMatrixActual = [cfsetting yactual];   % Storing CF setting and Yactual for that setting
varianc = (var(yactual'))';   % Finding the Variance for each CF setting
```

17.6

```
% Function Assumed Model takes inputs from RWH model, mean of Noises and
% the Covariance Matrix, which shows that Noise variables are independent.
% It then generates the Response for 200 Noise factors settings and finds
% the variance at each control factor setting for full model

% 03/24/2004 by Jagmeet Singh

function [ResponseMatrixAssumed, varianc] = Assumedmodel(bi, bij, bijk, MU, sigma)

global cfsetting w2 ncf nnf maxnoisevar;

nfsetting = lhsnorm(MU, sigma, maxnoisevar);
for cfruns = 1:128
   for nfruns = 1:maxnoisevar
      x(1,1:5)=nfsetting(nfruns,:);  % Defining Noise Factor Settings
      x(1,6:12)=cfsetting(cfruns,:);  % Defining CF Settings

      sumij=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            sumij=sumij+bij(i,j)*x(1,i)*x(1,j);
         end
      end

      sumijk=0;
      for i=1:(nnf+ncf)
         for j=i+1:(nnf+ncf)
            for k=j+1:(nnf+ncf)
               sumijk=sumijk + bijk(i,j,k)*x(1,i)*x(1,j)*x(1,k);
            end
         end
      end

      yassumed(cfruns,nfruns)=bi*x' + sumij + sumijk + normrnd(0,w2);
   end
end

ResponseMatrixAssumed = [cfsetting yassumed];  % Storing CF setting and Yassumed for that
setting
varianc = (var(yassumed'))';  % Finding the Variance for each CF setting
```

17.7

```
% Function to plot the outputs of the response

function output()

load variables;

% Plotting and Analysing Results from the runs

% Pie Chart when Std_actual is close to op_std_actual based on Optimal
% Stdev
figure; success=[(counter_per_dev_op) (maxmodels-counter_per_dev_op)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Success: ';'> 5%:    '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in reducing ST_d_e_v to 5% error (based on OP.STD_d_e_v) of OP.STD_d_e_v
for ',num2str(maxmodels),' models']);
hgsave('pie1');

      % Plotting histogram for per_dev_op
      figure;
      hist(per_dev_op);
      title(['Histogram for Percentage of (STD_a_c_t_u_a_l-
OP.STD_a_c_t_u_a_l)/OP.STD_a_c_t_u_a_l for ',num2str(maxmodels),' models']);
      colormap cool; iq = prctile(per_dev_op,[25 50 75]); tmax = max(hist(per_dev_op));
      line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
      line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile =
',num2str(iq(1))]...
         ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
      xlabel('((STD.Actual - OP.STD.Actual)/OP.STD.Actual) *100');
      ylabel(' number of systems ');
      hgsave('fig-pie1');

% Pie Chart when Std_actual is close to op_std_actual based on base
% Stdev
figure; success=[(counter_per_dev_base) (maxmodels-counter_per_dev_base)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Success: ';'> 5%:    '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in reducing ST_d_e_v to 5% error (based on Base.STD_a_c_t_u_a_l)
OP.STD_d_e_v for ',num2str(maxmodels),' models']);
hgsave('pie2');

% Pie Chart when OCF_Actual is same as OCF_Assumed
```

```
figure; success=[(counter_OCF) (maxmodels-counter_OCF)];
explode = [1 0]; colormap cool; hp = pie3(success,explode);
textobjs = findobj(hp,'Type','text'); oldstr = get(textobjs,{'String'});
Names = {'Same CF levels: ';'Diff CF levels: '}; newstr = strcat(Names, oldstr);
set(textobjs,{'String'},newstr);
pos = get(textobjs,{'Position'}); pos{1,:} = [ -0.28 -0.61 .35];
set(textobjs,{'Position'},pos);
title(['Success in prediction of OCF.Actual from OCF.Assumed ',num2str(maxmodels),' models']);
hgsave('pie3');

% Plotting Histograms
figure;
hist(std_fraction1);
title(['Histogram for Fraction of ST_d_e_v.Actual to STD.Base.Actual for ',num2str(maxmodels),'
models']);
colormap cool; iq = prctile(std_fraction1,[25 50 75]); tmax = max(hist(std_fraction1));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('STD.Actual/STD.Base.Actual');
ylabel(' number of systems ');
hgsave('fig1');

figure;
hist(std_fraction2);
title(['Histogram for Fraction of ST_d_e_v.Actual to OP.STD.Actual for ',num2str(maxmodels),'
models']);
colormap cool; iq = prctile(std_fraction2,[25 50 75]); tmax = max(hist(std_fraction2));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('STD.Actual/OP.STD.Actual');
ylabel(' number of systems ');
hgsave('fig2');

figure;
hist(std_fraction3);
title(['Histogram for Fraction of (STD_a_c_t_u_a_l-STD_b_a_s_e _a_c_t_u_a_l
_1)/(OP.STD_a_c_t_u_a_l-STD_b_a_s_e _a_c_t_u_a_l _1) for ',num2str(maxmodels),' models']);
colormap cool; iq = prctile(std_fraction3,[25 50 75]); tmax = max(hist(std_fraction3));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Actual-STD.Base.Actual1)/(OP.STD.Actual-STD.Base.Actual1)');
ylabel(' number of systems ');
hgsave('fig3');

figure;
hist(std_fraction4);
title(['Histogram for Fraction of (STD_a_c_t_u_a_l-STD_b_a_s_e
_a_c_t_u_a_l)/(OP.STD_a_c_t_u_a_l-STD_b_a_s_e _a_c_t_u_a_l) for ',num2str(maxmodels),'
models']);
```

```
colormap cool; iq = prctile(std_fraction4,[25 50 75]); tmax = max(hist(std_fraction4));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Actual-STD.Base.Actual)/(OP.STD.Actual-STD.Base.Actual)');
ylabel(' number of systems ');
hgsave('fig4');

figure;
hist(std_fraction5);
title(['Histogram for Fraction of STD_a_c_t_u_a_l/STD_b_a_s_e _a_c_t_u_a_l _1 for
',num2str(maxmodels),' models']);
colormap cool; iq = prctile(std_fraction5,[25 50 75]); tmax = max(hist(std_fraction5));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('(STD.Actual)/(STD.Base.Actual.1)');
ylabel(' number of systems ');
hgsave('fig5');

figure;
hist(improv_actual1);
title(['Histogram for % Improv. in ST_d_e_v(based on OCF.Assumed) for Actual Model for
',num2str(maxmodels),' models']);
colormap cool; iq = prctile(improv_actual1,[25 50 75]); tmax = max(hist(improv_actual1));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('((STD.Base.Actual - STD.Actual)/STD.Base.Actual)*100');
ylabel(' number of systems ');
hgsave('fig6');

figure;
hist(improv_actual2);
title(['Histogram for % Improv. in ST_d_e_v(based on OCF.Actual) for Actual Model for
',num2str(maxmodels),' models']);
colormap cool; iq = prctile(improv_actual2,[25 50 75]); tmax = max(hist(improv_actual2));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('((STD.Base.Actual - OP.STD.Actual)/STD.Base.Actual)*100');
ylabel(' number of systems ');
hgsave('fig7');

figure;
hist(improv_actual3);
title(['Histogram for % Improv. in ST_d_e_v(based on OCF.Assumed) for Actual Model for
',num2str(maxmodels),' models']);
colormap cool; iq = prctile(improv_actual3,[25 50 75]); tmax = max(hist(improv_actual3));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
```

```
xlabel('((STD.Base.Actual.1 - STD.Actual)/STD.Base.Actual.1) *100');
ylabel(' number of systems ');
hgsave('fig8');

figure;
hist(improv_actual4);
title(['Histogram for % Improv. in ST_d_e_v(based on OCF.Actual) for Actual Model for
',num2str(maxmodels),' models']);
colormap cool; iq = prctile(improv_actual4,[25 50 75]); tmax = max(hist(improv_actual4));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('((STD.Base.Actual.1 - OP.STD.Actual)/STD.Base.Actual.1) *100');
ylabel(' number of systems ');
hgsave('fig9');

figure;
hist(improv_assumed);
title(['Histogram for % Improv. in ST_d_e_v for Assumed Model for ',num2str(maxmodels),'
models']);
colormap cool; iq = prctile(improv_assumed,[25 50 75]); tmax = max(hist(improv_assumed));
line([iq(1) iq(1)],[0 tmax],'LineStyle','--'); line([iq(3) iq(3)],[0 tmax],'LineStyle','--','Color','r');
line([iq(2) iq(2)],[0 tmax],'LineWidth',2,'Color','m'); legend(['25_t_h Percentile = ',num2str(iq(1))]...
   ,['75_t_h Percentile = ',num2str(iq(3))],['Median = ',num2str(iq(2))],' Frequency');
xlabel('((STD.Base.Assumed - OP.STD.Assumed)/STD.Base.Assumed)*100');
ylabel(' number of systems ');
hgsave('fig10');

% Printing the results
sprintf(['mean(OP.STD.Actual) = ',num2str(mean(OP_STD_ACTUAL)),'  std(OP.STD.Actual) =
',num2str(std(OP_STD_ACTUAL))])
sprintf(['mean(STD.Actual) = ',num2str(mean(STD_ACTUAL)),'  std(STD.Actual) =
',num2str(std(STD_ACTUAL))])
sprintf([' (M[Std.Actual] - M[OP.STD.Actual])/M[OP.STD.Actual] * 100 = ',num2str(
((mean(STD_ACTUAL) - mean(OP_STD_ACTUAL))/mean(OP_STD_ACTUAL))*100),'%%'])
```